Author: tshingombe tshitadi

Curriculum

Career Discovery Application: Trade Electrical Experience Discovery in Data

Basic and advanced framework design course

⚡ Career Discovery Application: Trade Electrical Experience Discovery in Data

Course Type: Basic and Advanced Framework Design **Author**: Tshingombe Tshitadi

□ Scope

This course empowers learners to explore career pathways in the electrical trade through data-driven experience mapping, competency-based assessment, and digital portfolio development. It integrates technical inventory systems, logigram logic, and research methodologies to scaffold both foundational and advanced framework design.

□ Description

Learners will:

- Discover trade-specific competencies through guided exploration and diagnostic mapping
- Build digital portfolios using evidence-based learning artifacts
- Analyze career data to identify strengths, gaps, and growth trajectories
- Design and validate frameworks using logigram and inventory tools
- Apply research methods to evaluate educational and professional pathways

□ Overview

Module	Focus Area	Tools & Techniques
Career Mapping	Trade exposure, interest profiling	Diagnostic engines, career logs
Framework Design	Basic to advanced	VBA templates, modular

Module	Focus Area	Tools & Techniques
	structure	scaffolds
Data Analysis	Experience traceability	Rubric scoring, audit trails
Portfolio Development	Evidence integration	QR-coded logs, CPD formatting
Inventory Systems	Technical documentation	Competency indexing, asset tracking
Logigram Logic	Workflow visualization	Decision trees, process maps

Data Analysis

- **Techniques**:
  - Competency clustering
  - Portfolio traceability mapping
  - Rubric-based scoring
  - Predictive modeling for career progression
- **Outputs**:
  - Learner dashboards
  - Audit-ready documentation
  - Career trajectory visualizations

Research Methodic

- **Approach**:
  - Mixed-methods: qualitative interviews + quantitative portfolio analysis
  - Literature synthesis from NSQF, SDG, ISO, and vocational reform sources
  - Bibliographic mapping of standards, codes, and sectoral benchmarks

- **Tools**:
    - Logigram engines for workflow modeling
    - VBA-based audit trail generators
    - Evidence tagging and classification systems

✅ Advantages

- Modular, exportable frameworks
- Audit-ready documentation
- Integration with CPD and Expo formats
- Learner empowerment through data visibility
- Alignment with NSQF and SDG targets

⚠ Disadvantages

- Requires digital literacy and system access
- Initial setup may be resource-intensive
- Needs institutional buy-in for full deployment
- Complexity in multi-sector alignment

📦 Inventory & Logigram

- **Inventory**:
    - Skills, tools, and evidence artifacts indexed by domain
    - Mapped to NSQF levels and SDG outcomes
- **Logigram**:
    - Visual decision maps for career discovery
    - Assessment validation flows

- Portfolio export and submission logic

📚 Bibliographic Sources

🛠️ Electrical Trade Curriculum & Standards

1. **Curriculum and Assessment Policy Statement: Technical Occupational – Electrical Technology** Covers foundational and advanced electrical topics, assessment principles, and career pathways in South Africa.

2. 🇿🇦 **Occupational Certificate: Electrician – Curriculum Document** Details knowledge modules, practical skill modules, and work experience components aligned with LGSETA standards.

📊 Data Analysis, Portfolio & Assessment Systems

- **ISO/IEC Standards** – For electrical installations, safety, and system interoperability
- **NSQF Guidelines** – For competency-based curriculum mapping and modular framework design
- **UNESCO TVET Reports** – On vocational education reform, digital credentialing, and learner empowerment
- **IEEE Xplore & ScienceDirect** – For technical papers on logigram modeling, simulation tools, and audit trail systems
- **South African Qualifications Authority (SAQA)** – For qualification alignment and framework validation

Would you like me to help format this into a reference-ready section for CPD submission or Expo documentation? I can also scaffold it into APA or Harvard style if needed.

Table of Contents

⚡ Career Discovery Application: Trade Electrical Experience Discovery in Data.....2
 Scope.....2
 Description.....2
 Overview.....2
 Data Analysis.....3
 Research Methodic.....3
 Advantages.....3
⚠ Disadvantages.....3
 Inventory & Logigram.....3
 Bibliographic Sources.....4
 Electrical Trade Curriculum & Standards.....4
 Data Analysis, Portfolio & Assessment Systems.....4
VBA logigram and algorigram for messy workbooks: auto-audit, repair, and summarize.....14
What it does.....14
Module: mAuditEngine.....14
How to use.....22
Notes.....23
 Workbook Schema.....23
Sheet: `CompetencyMap`.....23
Sheet: `QualificationCriteria`.....23
Sheet: `Findings`.....24
⚙ VBA Engine: Logigram + Algorigram.....24
VBA logigram and algorigram for assessment framework, moderation, and SAQA mapping.....26
Workbook sheets.....26
VBA code.....28
What this gives you.....36
 Workbook Schema.....37
Sheet: SubjectMapping.....37

Sheet: Modules4IR 37
Sheet: EngineeringTheory 37
Sheet: EmbeddedSystems 38
Sheet: CurriculumProjects 38
Sheet: CareerPathways 38
⚙ **VBA Engine** 38
Workbook sheets to create 41
VBA code (paste into a standard module, e.g., mExpoAudit) 42
 Interpretation of Your Structure 49
 Curriculum Matrix 49
 VBA UserForm Skeleton 49
 Suggested VBA Logigram Architecture 50
1. Data Mapping Module 50
 Visualization Strategy 50
 Next Steps 50
 Conceptual Architecture: Logigram + Algorigram Engine 51
 VBA Implementation Blueprint 51
1. Data Structure Initialization 51
5. UserForm Dashboard 52
 Strategic Extensions 52
VBA logigram and algorigram engine for training, system specs, and courses 53
Overview 53
Worksheet assumptions 53
UserForm with TreeView + ListView 59
Built-in algorigram rules included 61
How to run 62
Want me to tailor this to your exact paste? 62
 Logigram Structure: Academic Progression Engine 62
 VBA Implementation Blueprint 63
1. Data Normalization 63
2. Logigram Node Generator 63
4. Irregularity Tracker 63
5. UserForm Dashboard 63
 Sample Output Summary 63

- **Institutional Compliance Logigram** 64
- **Next Steps** 64
- Conceptual Architecture 64
- VBA Implementation Blueprint 65
  1. Data Structure Initialization 65
  2. Logigram Node Generator 65
  5. UserForm Dashboard 66
  - Strategic Extensions 66
  - Conceptual Breakdown 67
    - Logigram: Curriculum Structure 67
  - **Algorigram: Progress Logic** 67
- **VBA Implementation Blueprint** 67
  1. Data Sheet Setup 67
  2. Logigram Node Generator 68
- UserForm Dashboard (Suggested Layout) 69
- Strategic Extensions 69
- Conceptual Architecture 69
- VBA Implementation Blueprint 70
  1. Data Sheet Setup 70
  2. Logigram Node Generator 70
  5. UserForm Dashboard 71
- Strategic Extensions 71

VBA logigram and algorigram for ION8650 wiring and DOL starter logic 72
- Overview 72
- Key rules encoded 72
  ION8650/8600, Form 35/35S, 4-Wire WYE, 2 PTs, 3 CTs (Volts Mode = DELTA) . 72
  DOL starter variants (contactor + overload) 72
  TeSys T LTMR (installation guide anchors) 73
- Workbook setup 73
- Class for nodes 73
- UserForm (TreeView + ListView) 77
- How to use 79
- 7. Conceptual Architecture 79
- 8. VBA Implementation Blueprint 80

9. 1. Data Sheet Setup....80
4. Distance Validator....81
 Strategic Extensions....82
VBA logigram and algorigram for LTMR RJ45, SoMove connections, and parameter registry....82
Workbook setup....82
Data model classes....83
UserForm for navigation and validation....89
Quick actions you can add....90
 Logigram Structure....91
 VBA Implementation Blueprint....91
1. Data Sheet Setup....91
2. Validation Function....92
5. UserForm Dashboard....93
 Sample Output....93
Strategic Extensions....93
VBA logigram and algorigram for APC Matrix-UPS diagnostics....93
Workbook structure....93
UPS_Specs....94
UPS_Status....94
UPS_Options....94
Node model and engine....95
Class: cNode....95
Built-in calculations and checks....100
How to run....100
Optional upgrades....101
VBA logigram and algorigram for Matrix-UPS diagnostics and setup....101
Workbook schema....101
UPS_Specs....101
UPS_Status....102
UPS_Options....102
Classes....102
Quick start....108
VBA logigram and algorigram for access-layer design and oversubscription....109
Workbook schema....109

Switches..... 109
Loads..... 109
StackPlan..... 109
PoEProfiles..... 109
What this engine does..... 110
Class: cNode..... 110
UserForm: frmAccess..... 114
 Notes and extensions..... 116
VBA logigram and algorigram for switch stack design, oversubscription, and uplink planning..... 116
Workbook schema..... 117
Switches..... 117
Loads..... 117
StackPlan..... 117
Optics..... 117
FiberPlant..... 118
What the engine computes..... 118
Class model..... 118
Core engine..... 118
 Workbook Schema..... 118
Topology..... 119
UplinkMatrix..... 119
SecurityFeatures..... 119
 Logigram + Algorigram VBA Engine..... 119
Class: cNode..... 119
 Navigation UserForm: `frmCampusLAN`..... 122
VBA logigram and algorigram for high availability and wireless controller design. 122
Workbook schema..... 123
HA_Features..... 123
WLAN_Controllers..... 123
WLAN_Design..... 124
Class model..... 124
Core engine: logigram + algorigram..... 124
What you get..... 130

VBA logigram and algorigram for campus platform selection and wireless capabilities....130
Workbook schema....131
Platform_Tiers....131
AP_Capabilities....131
WLC_Profiles....132
Sites....132
Glossary....132
Class model....132
What this gives you....141
Workbook schema....141
DocMeta....141
EnvParams....141
Architectures....142
PhysicalEnv....142
Devices....142
ComplianceRules....143
Class model....143
Built-in checks and findings....151
Fast start....151
Reusable VBA logigram and algorigram scaffold....151
1) Workbook schema....152
Sheet: Nodes....152
Sheet: NodeMeta....152
Sheet: Rules....152
2) Class: cNode....153
3) Engine: mLogiAlgo....153
4) UserForm: frmLogiAlg....159
5) How to run....160
6) Extend in seconds....161
 Step 1: Class Module – cConcept....161
 Step 3: UserForm – frmConceptViewer....164
 How to Run....166
VBA logigram and algorigram for digital logic and curriculum mapping....166
1) Workbook sheets....166

Sheet: LogicRules....166
Sheet: LogicInputs....166
Sheet: TruthTable....167
Sheet: Curriculum....167
Sheet: CurriculumFindings....167
2) Class: cNode (for curriculum logigram)....167
6) Notes and extensions....176
VBA logigram and algorigram for banking automation, circuit design, and SARB alignment....176
Workbook sheets to create....176
Sheet: Domains....176
Sheet: CircuitDesign....177
Sheet: SARB_Applications....177
Sheet: EducationAlignment....177
Sheet: CareerOutcomes....177
Sheet: ProjectTakeaways....177
VBA code (paste into a standard module, e.g., mBankingFramework)....178
How to run....183
Optional extensions....184
Workbook sheets to create....184
VBA code (paste into a standard module, e.g., mNatFramework)....185
What you get....194
VBA logigram and algorigram for school management and vocational guidance...194
Workbook sheets to create....194
VBA code (paste into a standard module, e.g., mSchoolVocFramework)....195
How to run....200
 Workbook Schema....200
Sheet: EvidenceArtifacts....200
Sheet: WindPowerLog....201
Sheet: SolarPanelLog....201
Sheet: InsulatorLog....201
Sheet: ConductorCapacityLog....202
Sheet: JumperInstallationLog....202
Sheet: RelayTestLog....202
Sheet: MaintenanceLog....202

⚙ VBA Engine (paste into a standard module, e.g., mRenewableAudit)..... 203
VBA logigram and algorigram for OOA/UML crime management system..... 205
Workbook sheets to create..... 206
VBA code (paste into a standard module, e.g., mOOA_Audit)..... 207
What you get..... 213
 Core Modules and VBA Logic Blocks..... 213
 Sample VBA Snippet: Chain of Custody Tracker..... 214
 Tools You Can Integrate..... 215
 VBA Logigramm & Algorigramm Engine for Crime Resolution..... 215
 Core Architecture..... 215
 1. Double Integral Simulation (Area Density)..... 215
 2. Collision Scenario Analyzer..... 216
 3. Evidence Decay Model..... 216
 Deployment Strategy..... 217
Workbook structure and named ranges..... 217
Sheets..... 217
Named ranges..... 218
Logigram rules and algorigram flows..... 218
Core VBA modules..... 218
2) Course logigram: sequencing and status..... 219
Buttons and quick wiring..... 224
ow to use and tailor..... 225
Optional extensions..... 225
Workbook schema and setup..... 225
Safety and readiness logigram..... 226
Mechanics algorigram: beams, frames, calibration, and validation..... 228
Session orchestration, measurement logging, and reporting..... 230
Buttons and quick wiring..... 234
Extending to electronic/audiometer labs and procurement..... 234
VBA logigram and algorigram for DL NGL lab integration (BASE, STUDENT, CIMSIM, IoT, DATA, CYBER)..... 235
Workbook schema..... 235
Logigram rules..... 235
Algorigram flows..... 236
Core VBA..... 236

Utilities and config.....236
Readiness logigram.....237
Scenario lifecycle.....239

VBA logigram and algorigram for messy workbooks: auto-audit, repair, and summarize

This drop-in VBA program scans your workbook, detects common data blocks (sales tables, loan name manager, outline stats, orders/customers, schedules, inventory, energy logs), fixes errors, fills formulas, and writes an auditable Findings log. It's modular: each "Fix_" routine is a node in the logigram; each test/repair produces algorigram findings (PASS/FAIL/WARN).

What it does

- Sales table: fixes header typos (QUATITY→QUANTITY), computes Subtotal, Discount, Total, clears stray ")" and #VALUE!.
- Name Manager (loan): validates interest, amount, months, payment; computes PMT and variance.
- Outline stats: calculates Max, 90th percentile, Median.
- Product inventory: computes value = Unit Price × Units in Stock and produces count/avg/sum.
- Orders/Customers: basic joins sanity checks and report skeleton.
- Schedule (book production): fills WORKDAYS from Start Date and Working Days Budget.
- Energy log: computes VA, W, kWh (if period present), and Total Cost if tariff named range exists.
- Global audit: flags #NAME?, #VALUE!, unmatched parentheses, empty required cells; writes "Findings" sheet.

Module: mAuditEngine

Paste into a standard module. Requires "Microsoft Scripting Runtime" reference.

vba

```
' Module: mAuditEngine
Option Explicit

' Findings row pointer
```

```
Private gFindRow As Long

Public Sub Run_Audit_And_Fix()
    Application.ScreenUpdating = False
    On Error GoTo done

    InitFindings

    ' 1) Sales table repair (Quantity/PriceEach/Subtotal/Discount/Total)
    Fix_SalesTables

    ' 2) Validate loan Name Manager block
    Fix_LoanNames

    ' 3) Outline stats (Max, P90, Median)
    Fix_OutlineStats

    ' 4) Product inventory and simple analysis
    Fix_Inventory

    ' 5) Orders / Customers sanity + report header
    Fix_OrdersCustomers

    ' 6) Schedule (simple book production WORKDAYS)
    Fix_Schedule

    ' 7) Energy log computations
    Fix_EnergyLog

    ' 8) Global scan for errors/artifacts
    Audit_GlobalErrors

done:
    Application.ScreenUpdating = True
    MsgBox "Audit complete. See 'Findings' sheet.", vbInformation
End Sub

' ================== Findings ==================

Private Sub InitFindings()
    Dim ws As Worksheet
    On Error Resume Next
    Application.DisplayAlerts = False
    Worksheets("Findings").Delete
    Application.DisplayAlerts = True
    On Error GoTo 0

    Set ws = Worksheets.Add(After:=Worksheets(Worksheets.Count))
    ws.Name = "Findings"
    ws.Range("A1:E1").Value = Array("Area", "Sheet", "Issue", "Detail",
"Action")
    gFindRow = 1
End Sub

Private Sub AddFinding(area$, sheetName$, issue$, detail$, action$)
```

```
    Dim ws As Worksheet: Set ws = Worksheets("Findings")
    gFindRow = gFindRow + 1
    ws.Cells(gFindRow, 1).Value = area
    ws.Cells(gFindRow, 2).Value = sheetName
    ws.Cells(gFindRow, 3).Value = issue
    ws.Cells(gFindRow, 4).Value = detail
    ws.Cells(gFindRow, 5).Value = action
End Sub

' ================== 1) Sales tables ==================

Private Sub Fix_SalesTables()
    Dim ws As Worksheet
    For Each ws In ThisWorkbook.Worksheets
        Dim hdrR As Long, hdrC As Long
        hdrR = FindHeaderRow(ws, Array("QUATITY", "QUANTITY", "PRICE EACH",
"SUBTOTAL", "DISCOUNT", "TOTAL"), hdrC)
        If hdrR > 0 Then
            Dim rngHdr As Range: Set rngHdr = ws.Rows(hdrR)
            ' Normalize headers
            NormalizeHeader ws, hdrR, "QUATITY", "QUANTITY"
            NormalizeHeader ws, hdrR, "PRICE EACH", "PRICE EACH"
            NormalizeHeader ws, hdrR, "SUBTOTAL", "SUBTOTAL"
            NormalizeHeader ws, hdrR, "DISCOUNT", "DISCOUNT"
            NormalizeHeader ws, hdrR, "TOTAL", "TOTAL"

            Dim cQty&, cPrice&, cSub&, cDisc&, cTot&
            cQty = FindCol(ws, hdrR, "QUANTITY")
            cPrice = FindCol(ws, hdrR, "PRICE EACH")
            cSub = FindCol(ws, hdrR, "SUBTOTAL")
            cDisc = FindCol(ws, hdrR, "DISCOUNT")
            cTot = FindCol(ws, hdrR, "TOTAL")

            If cQty * cPrice * cSub * cTot = 0 Then
                AddFinding "Sales", ws.Name, "Missing required column(s)",
"QUANTITY/PRICE EACH/SUBTOTAL/TOTAL", "Review headers"
            Else
                Dim r&, lastR&
                lastR = ws.Cells(ws.Rows.Count, cQty).End(xlUp).Row
                For r = hdrR + 1 To lastR
                    Dim vQty, vPrice
                    vQty = ws.Cells(r, cQty).Value
                    vPrice = ws.Cells(r, cPrice).Value

                    ' Clean stray ")" and error values
                    CleanCell ws.Cells(r, cSub)
                    CleanCell ws.Cells(r, cTot)

                    If IsNumeric(vQty) And IsNumeric(vPrice) Then
                        ws.Cells(r, cSub).Value = CDbl(vQty) * CDbl(vPrice)
                        ' Optional discount: if blank, assume 0
                        Dim vDisc: vDisc = 0
                        If cDisc > 0 Then
                            If IsNumeric(ws.Cells(r, cDisc).Value) Then vDisc
= CDbl(ws.Cells(r, cDisc).Value)
```

```
                        End If
                        ws.Cells(r, cTot).Value = ws.Cells(r, cSub).Value -
vDisc
                    ElseIf Len(vQty) = 0 And Len(vPrice) = 0 Then
                        ' End of data row set, skip
                    Else
                        AddFinding "Sales", ws.Name, "#VALUE! in row", "Row "
& r & " qty/price non-numeric", "Correct inputs"
                    End If
                Next r
                AddFinding "Sales", ws.Name, "Computed", "Subtotal/Total
recalculated", "OK"
            End If
        End If
    Next ws
End Sub

Private Sub CleanCell(ByVal c As Range)
    If IsError(c.Value) Then c.ClearContents
    If Trim$(CStr(c.Value)) = ")" Then c.ClearContents
End Sub

Private Sub NormalizeHeader(ws As Worksheet, hdrRow&, fromLbl$, toLbl$)
    Dim col&: col = FindCol(ws, hdrRow, fromLbl$)
    If col > 0 Then ws.Cells(hdrRow, col).Value = toLbl$
End Sub

' ================== 2) Loan name manager block ==================

Private Sub Fix_LoanNames()
    On Error GoTo safeExit
    Dim i As Double, p As Double, n As Long, pay As Double
    i = CDbl(Evaluate("INTEREST"))
    p = CDbl(Evaluate("LOAN AMOUNT"))
    n = CLng(Evaluate("MONTH"))
    pay = CDbl(Evaluate("PAYMENT"))

    Dim rate As Double: rate = i / 12
    Dim pmt As Double
    If rate <> 0 Then
        pmt = -WorksheetFunction.Pmt(rate, n, p)
    Else
        pmt = -(p / n)
    End If
    Dim diff As Double: diff = pay - pmt
    AddFinding "Loan", "(Names)", "PMT check", "Named PAYMENT=" & Format(pay,
"0.00") & " vs PMT=" & Format(pmt, "0.00"), IIf(Abs(diff) < 0.01, "OK",
"Adjust PAYMENT"))
safeExit:
End Sub

' ================== 3) Outline stats ==================

Private Sub Fix_OutlineStats()
    Dim ws As Worksheet
```

```
    For Each ws In ThisWorkbook.Worksheets
        Dim r0&, c0&: r0 = FindHeaderRow(ws, Array("DAYS WITH A", "DAYS WAS
GOOD", "MAXIMUN", "90 TH PERCENTILE", "MEDIAN"), c0)
        If r0 > 0 Then
            Dim lastR&: lastR = ws.Cells(ws.Rows.Count, c0).End(xlUp).Row
            ' Assume data in first two columns under those headers
            Dim dataRng As Range: Set dataRng = ws.Range(ws.Cells(r0 + 1,
c0), ws.Cells(lastR, c0))
            If WorksheetFunction.CountA(dataRng) > 0 Then
                ' Where to place outputs: find columns labeled
                Dim cMax&, cP90&, cMed&
                cMax = FindCol(ws, r0, "MAXIMUN")
                cP90 = FindCol(ws, r0, "90 TH PERCENTILE")
                cMed = FindCol(ws, r0, "MEDIAN")
                If cMax * cP90 * cMed > 0 Then
                    ws.Cells(r0 + 1, cMax).Value =
WorksheetFunction.Max(dataRng)
                    ws.Cells(r0 + 1, cP90).Value =
WorksheetFunction.Percentile_Exc(dataRng, 0.9)
                    ws.Cells(r0 + 1, cMed).Value =
WorksheetFunction.Median(dataRng)
                    AddFinding "Outline", ws.Name, "Stats computed",
"Max/P90/Median", "OK"
                Else
                    AddFinding "Outline", ws.Name, "Missing output headers",
"MAXIMUN / 90TH PERCENTILE / MEDIAN", "Label columns"
                End If
            End If
        End If
    Next ws
End Sub

' ================== 4) Inventory analysis ==================

Private Sub Fix_Inventory()
    Dim ws As Worksheet
    For Each ws In ThisWorkbook.Worksheets
        Dim r0&, c0&: r0 = FindHeaderRow(ws, Array("PRODUCT ID", "UNITY
PRICE", "UNIT PRICE", "VALUE OF INVENTORY", "UNITS STOCK"), c0)
        If r0 > 0 Then
            Dim cPID&, cPrice&, cUnits&, cValue&
            cPID = FindCol(ws, r0, "PRODUCT ID")
            cPrice = FindColAny(ws, r0, Array("UNITY PRICE", "UNIT PRICE"))
            cUnits = FindColAny(ws, r0, Array("UNITS STOCK", "UNITS IN
STOCK"))
            cValue = FindColAny(ws, r0, Array("VALUE OF INVENTORY", "VALUE OF
INVENTORY UNITS STOCK"))
            If cPrice * cUnits > 0 Then
                Dim lastR&: lastR = ws.Cells(ws.Rows.Count,
cPrice).End(xlUp).Row
                Dim r&
                For r = r0 + 1 To lastR
                    If IsNumeric(ws.Cells(r, cPrice).Value) And
IsNumeric(ws.Cells(r, cUnits).Value) Then
```

```
                        If cValue = 0 Then cValue = cUnits + 1: ws.Cells(r0,
cValue).Value = "VALUE OF INVENTORY"
                        ws.Cells(r, cValue).Value = CDbl(ws.Cells(r,
cPrice).Value) * CDbl(ws.Cells(r, cUnits).Value)
                    End If
                Next r
                AddFinding "Inventory", ws.Name, "Computed", "Inventory value
calculated", "OK"
            Else
                AddFinding "Inventory", ws.Name, "Missing columns", "Unit
Price / Units Stock", "Fix headers"
            End If
        End If
    Next ws
End Sub

' ================== 5) Orders / Customers ==================

Private Sub Fix_OrdersCustomers()
    Dim wsO As Worksheet, wsC As Worksheet
    Set wsO = FindSheetByHeaders(Array("ORDER ID", "CUSTOMER ID", "EMPLOYEER
ID", "ORDER DATE"))
    Set wsC = FindSheetByHeaders(Array("FIST NAME", "FIRST NAME", "LAST
NAME", "CUSTOMERS", "CUSTOMER"))
    If wsO Is Nothing Or wsC Is Nothing Then Exit Sub

    ' Normalize first/last name headers
    Dim rC&, tmp&
    rC = FindHeaderRow(wsC, Array("FIST NAME", "FIRST NAME", "LAST NAME"),
tmp)
    NormalizeHeader wsC, rC, "FIST NAME", "FIRST NAME"

    AddFinding "Orders/Customers", wsO.Name & "/" & wsC.Name, "Sanity",
"Tables detected", "OK"

    ' Create a basic report header sheet if not present
    Dim wsR As Worksheet
    Set wsR = GetOrCreate("Report_Customers")
    wsR.Cells.Clear
    wsR.Range("A1:E1").Value = Array("CUSTOMER ID", "FIRST NAME", "LAST
NAME", "ORDERS COUNT", "LAST ORDER DATE")
    ' You can extend with a real join if consistent IDs exist.
End Sub

' ================== 6) Schedule (book production) ==================

Private Sub Fix_Schedule()
    Dim ws As Worksheet
    For Each ws In ThisWorkbook.Worksheets
        If InStr(1, UCase$(ws.UsedRange.Cells(1, 1).Value), "SIMPLE BOOK
PRODUCT SCHEDULE", vbTextCompare) > 0 Then
            ' Find START DATE and WORKING DAYS BUDGET rows, write WORKDAYS
labels and dates
            Dim rStart&, rBudget&
            rStart = FindRowContains(ws, "START DATE")
```

```
            rBudget = FindRowContains(ws, "WORKIG DAYS BUDGET")
            If rStart > 0 And rBudget > 0 Then
                Dim startDate As Variant: startDate = NextNumericRight(ws,
rStart)
                Dim workDays As Variant: workDays = NextNumericRight(ws,
rBudget)
                If IsDate(startDate) And IsNumeric(workDays) Then
                    Dim endDate As Date
                    endDate = WorksheetFunction.WorkDay(startDate,
CLng(workDays))
                    AddFinding "Schedule", ws.Name, "Plan", "Start=" &
CDate(startDate) & " Workdays=" & CLng(workDays) & " End=" & endDate, "OK"
                Else
                    AddFinding "Schedule", ws.Name, "Missing values", "Start
Date or Working Days Budget not numeric/date", "Fill inputs"
                End If
            End If
        End If
    Next ws
End Sub

' ================= 7) Energy log =================

Private Sub Fix_EnergyLog()
    Dim ws As Worksheet
    For Each ws In ThisWorkbook.Worksheets
        Dim r0&, c0&: r0 = FindHeaderRow(ws, Array("UNIT", "CHARGE",
"CURRENT", "QUATITY AH", "QUANTITY AH", "VOLTAGE", "VOLT AMP", "WATH",
"WATT", "COS", "KWH", "MONTH", "TOTAL COST"), c0)
        If r0 > 0 Then
            ' Normalize typos
            NormalizeHeader ws, r0, "QUATITY AH", "QUANTITY AH"
            NormalizeHeader ws, r0, "WATH", "WATT"

            Dim cI&, cV&, cVA&, cW&, cPF&, cKWh&, cCost&
            cI = FindColAny(ws, r0, Array("CURRENT"))
            cV = FindColAny(ws, r0, Array("VOLTAGE"))
            cVA = FindColAny(ws, r0, Array("VOLT AMP", "VA"))
            cW = FindColAny(ws, r0, Array("WATT", "W"))
            cPF = FindColAny(ws, r0, Array("COS", "POWER FACTOR"))
            cKWh = FindColAny(ws, r0, Array("KWH"))
            cCost = FindColAny(ws, r0, Array("TOTAL COST"))

            Dim lastR&: lastR = ws.Cells(ws.Rows.Count, cV).End(xlUp).Row
            Dim r&
            For r = r0 + 1 To lastR
                If cV * cI > 0 Then
                    Dim vV, vI, vPF
                    vV = ws.Cells(r, cV).Value
                    vI = ws.Cells(r, cI).Value
                    vPF = IIf(cPF > 0, ws.Cells(r, cPF).Value, 1)
                    If IsNumeric(vV) And IsNumeric(vI) Then
                        If cVA = 0 Then cVA = cV + 1: ws.Cells(r0, cVA).Value
= "VOLT AMP"
                        ws.Cells(r, cVA).Value = CDbl(vV) * CDbl(vI)
```

```
                        If cW = 0 Then cW = cVA + 1: ws.Cells(r0, cW).Value =
"WATT"
                        ws.Cells(r, cW).Value = ws.Cells(r, cVA).Value *
IIf(IsNumeric(vPF), CDbl(vPF), 1)
                    End If
                End If
            Next r

            ' Cost if tariff exists as Name 'TARIFF_PER_KWH'
            On Error Resume Next
            Dim tariff As Double: tariff = CDbl(Evaluate("TARIFF_PER_KWH"))
            On Error GoTo 0
            If cKWh > 0 And cCost > 0 And tariff > 0 Then
                For r = r0 + 1 To lastR
                    If IsNumeric(ws.Cells(r, cKWh).Value) Then
                        ws.Cells(r, cCost).Value = CDbl(ws.Cells(r,
cKWh).Value) * tariff
                    End If
                Next r
            End If
            AddFinding "Energy", ws.Name, "Computed", "VA/W (and Cost if
tariff set) calculated", "OK"
        End If
    Next ws
End Sub

' ================== 8) Global error scan ==================

Private Sub Audit_GlobalErrors()
    Dim ws As Worksheet
    For Each ws In ThisWorkbook.Worksheets
        Dim rng As Range: Set rng = ws.UsedRange
        If rng Is Nothing Then GoTo NextWs
        Dim c As Range
        For Each c In rng
            If IsError(c.Value) Then
                AddFinding "Global", ws.Name, "Cell error", c.Address(0, 0) &
" = " & CStr(c.Text), "Investigate"
            ElseIf Trim$(CStr(c.Value)) = ")" Then
                AddFinding "Global", ws.Name, "Stray parenthesis",
c.Address(0, 0), "Cleared"
                c.ClearContents
            End If
        Next c
NextWs:
    Next ws
End Sub

' ================== Helpers ==================

Private Function FindHeaderRow(ws As Worksheet, headers As Variant, ByRef
firstCol&) As Long
    Dim r&, maxR&: maxR = Application.Min(50, ws.UsedRange.Rows.Count)
    Dim h As Variant, c As Range
    For r = 1 To maxR
```

```
        For Each h In headers
            Set c = RowFind(ws, r, CStr(h))
            If Not c Is Nothing Then firstCol = c.Column: FindHeaderRow = r:
Exit Function
        Next h
    Next r
End Function

Private Function RowFind(ws As Worksheet, row&, text$) As Range
    Dim rng As Range: Set rng = ws.Rows(row)
    Dim f As Range
    Set f = rng.Find(What:=text, LookIn:=xlValues, LookAt:=xlPart,
MatchCase:=False)
    If Not f Is Nothing Then Set RowFind = f
End Function

Private Function FindCol(ws As Worksheet, hdrRow&, header$) As Long
    Dim f As Range
    Set f = ws.Rows(hdrRow).Find(What:=header, LookIn:=xlValues,
LookAt:=xlWhole, MatchCase:=False)
    If Not f Is Nothing Then FindCol = f.Column
End Function

Private Function FindColAny(ws As Worksheet, hdrRow&, headers As Variant) As
Long
    Dim h As Variant
    For Each h In headers
        FindColAny = FindCol(ws, hdrRow, CStr(h))
        If FindColAny > 0 Then Exit Function
    Next h
End Function

Private Function FindSheetByHeaders(headers As Variant) As Worksheet
    Dim ws As Worksheet, tmp&
    For Each ws In ThisWorkbook.Worksheets
        If FindHeaderRow(ws, headers, tmp) > 0 Then Set FindSheetByHeaders =
ws: Exit Function
    Next ws
End Function

Private Function FindRowContains(ws As Worksheet, text$) As Long
    Dim r&, maxR&: maxR = Application.Min(200, ws.UsedRange.Rows.Count)
    For r = 1 To maxR
        If InStr(1, UCase$
(Join(Application.Transpose(Application.Transpose(ws.Rows(r).Value)), " ")),
UCase$(text), vbTextCompare) > 0 Then
            FindRowContains = r: Exit Function
        End If
    Next r
End Function

Private Function NextNumericRight(ws As Worksheet, row&) As Variant
    Dim lastC&: lastC = ws.Cells(row, ws.Columns.Count).End(xlToLeft).Column
    Dim c&
    For c = 1 To lastC
```

```
        If IsDate(ws.Cells(row, c).Value) Or IsNumeric(ws.Cells(row,
c).Value) Then
            NextNumericRight = ws.Cells(row, c).Value
            Exit Function
        End If
    Next c
End Function

Private Function GetOrCreate(name$) As Worksheet
    On Error Resume Next
    Set GetOrCreate = Worksheets(name)
    On Error GoTo 0
    If GetOrCreate Is Nothing Then
        Set GetOrCreate = Worksheets.Add(After:=Worksheets(Worksheets.Count))
        GetOrCreate.Name = name
    End If
End Function
```

How to use

1. Open the VBA editor (Alt+F11), insert a new module, paste the code.
2. Tools > References… tick “Microsoft Scripting Runtime”.
3. Run the macro: Run_Audit_And_Fix.
4. Review “Findings” sheet for an auditable list of issues, fixes, and next actions.

Notes

Here's a VBA-powered logigram and algorigram engine tailored to your Core Competency Areas in engineering training, assessment, and qualification mapping. This framework builds a hierarchical map of domains, skills, and assessment methods (logigram), and runs rule-based checks to validate completeness, alignment, and integrity (algorigram).

□ Workbook Schema

Create these sheets with exact headers:

Sheet: `CompetencyMap`

Domain	SubArea	Skill/Topic	Evidence Required	Assessment Method
Installation &	Wiring	Mounting &	Labeled cables,	Practical task,

Domain	SubArea	Skill/Topic	Evidence Required	Assessment Method
Wiring		wiring control equipment	terminal numbering, schematic interpretation	supervisor sign-off
Installation & Wiring	Wiring	Cable labeling & sizing	Wire gauge, voltage rating, insulation type	Lab test, documentation review
Technical Drawing & Documentation	Drawings	Base Assembly Drawing	Identify work relationships	Correct interpretation, clarity
Diagnostics & Maintenance	Faults	Fault diagnosis	Error codes, schematic tracing	Fault report, simulation
Material Science & Testing	Heat Transfer	$Q = m \cdot c \cdot \Delta T$	Thermal diagnostics	Energy audit

Sheet: `QualificationCriteria`

Element	Details
Performance Package	Evidence of installation, labeling, diagnostics, and documentation
Quality Plan	Final inspection, random checks, acceptance criteria
Assessment Tools	Logbooks, test reports, schematic interpretation, fault tracing
Integrity Body	Responsible for validation, verification, and certification
Credit Mapping	Aligns with NQF, SAQA, ISAT, and QCTO standards

Sheet: `Findings`

Leave empty; the code will populate it with logigram and algorigram results.

⚙ VBA Engine: Logigram + Algorigram

Paste this into a standard module named `mCompetencyEngine`:

```
Option Explicit

Public Sub BuildCompetencyLogigram()
    Dim ws As Worksheet: Set ws = ThisWorkbook.Sheets("CompetencyMap")
    Dim wsF As Worksheet: Set wsF = GetOrCreate("Findings")
    wsF.Cells.Clear
    wsF.Range("A1:D1").Value = Array("Level", "Item", "Issue", "Detail")

    Dim lastRow As Long: lastRow = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    Dim r As Long, rowF As Long: rowF = 1

    Dim domain$, subarea$, skill$, evidence$, assess$
    Dim domainSet As Object: Set domainSet =
CreateObject("Scripting.Dictionary")
    Dim subareaSet As Object: Set subareaSet =
CreateObject("Scripting.Dictionary")

    For r = 2 To lastRow
        domain = Trim(ws.Cells(r, 1).Value)
        subarea = Trim(ws.Cells(r, 2).Value)
        skill = Trim(ws.Cells(r, 3).Value)
        evidence = Trim(ws.Cells(r, 4).Value)
        assess = Trim(ws.Cells(r, 5).Value)

        If Len(domain) = 0 Then
            rowF = rowF + 1
            wsF.Cells(rowF, 1).Value = "Domain"
            wsF.Cells(rowF, 2).Value = "(Row " & r & ")"
            wsF.Cells(rowF, 3).Value = "Missing domain"
            wsF.Cells(rowF, 4).Value = "Fill domain name"
        Else
            domainSet(domain) = True
        End If

        If Len(subarea) = 0 Then
            rowF = rowF + 1
            wsF.Cells(rowF, 1).Value = "SubArea"
            wsF.Cells(rowF, 2).Value = skill
            wsF.Cells(rowF, 3).Value = "Missing subarea"
            wsF.Cells(rowF, 4).Value = "Categorize skill under subarea"
        Else
            subareaSet(subarea) = True
        End If

        If Len(skill) = 0 Then
            rowF = rowF + 1
            wsF.Cells(rowF, 1).Value = "Skill"
            wsF.Cells(rowF, 2).Value = "(Row " & r & ")"
            wsF.Cells(rowF, 3).Value = "Missing skill/topic"
            wsF.Cells(rowF, 4).Value = "Specify competency item"
        End If

        If Len(evidence) = 0 Then
            rowF = rowF + 1
```

```
            wsF.Cells(rowF, 1).Value = "Evidence"
            wsF.Cells(rowF, 2).Value = skill
            wsF.Cells(rowF, 3).Value = "Missing evidence"
            wsF.Cells(rowF, 4).Value = "Define what proves competency"
        End If

        If Len(assess) = 0 Then
            rowF = rowF + 1
            wsF.Cells(rowF, 1).Value = "Assessment"
            wsF.Cells(rowF, 2).Value = skill
            wsF.Cells(rowF, 3).Value = "Missing assessment method"
            wsF.Cells(rowF, 4).Value = "Specify how skill is tested"
        End If
    Next r

    ' Summary counts
    rowF = rowF + 2
    wsF.Cells(rowF, 1).Value = "Summary"
    wsF.Cells(rowF, 2).Value = "Domains"
    wsF.Cells(rowF, 3).Value = domainSet.Count
    rowF = rowF + 1
    wsF.Cells(rowF, 2).Value = "SubAreas"
    wsF.Cells(rowF, 3).Value = subareaSet.Count
    rowF = rowF + 1
    wsF.Cells(rowF, 2).Value = "Skills Mapped"
    wsF.Cells(rowF, 3).Value = lastRow - 1

    wsF.Columns.AutoFit
End Sub

Public Sub ValidateQualificationCriteria()
    Dim wsQ As Worksheet: Set wsQ =
ThisWorkbook.Sheets("QualificationCriteria")
    Dim wsF As Worksheet: Set wsF = GetOrCreate("Findings")
    Dim lastRow As Long: lastRow = wsQ.Cells(wsQ.Rows.Count, 1).End(xlUp).Row
    Dim r As Long, rowF As Long: rowF = wsF.Cells(wsF.Rows.Count,
1).End(xlUp).Row + 1

    Dim elem$, detail$
    For r = 2 To lastRow
        elem = Trim(wsQ.Cells(r, 1).Value)
        detail = Trim(wsQ.Cells(r, 2).Value)

        If Len(elem) = 0 Then
            wsF.Cells(rowF, 1).Value = "Qualification"
            wsF.Cells(rowF, 2).Value = "(Row " & r & ")"
            wsF.Cells(rowF, 3).Value = "Missing element"
            wsF.Cells(rowF, 4).Value = "Fill qualification element name"
            rowF = rowF + 1
        End If

        If Len(detail) = 0 Then
            wsF.Cells(rowF, 1).Value = "Qualification"
            wsF.Cells(rowF, 2).Value = elem
            wsF.Cells(rowF, 3).Value = "Missing detail"
```

```
            wsF.Cells(rowF, 4).Value = "Describe qualification criteria"
            rowF = rowF + 1
        End If
    Next r
    wsF.Columns.AutoFit
End Sub

Private Function GetOrCreate(name$) As Worksheet
    On Error Resume Next
    Set GetOrCreate = Worksheets(name)
    On Error GoTo 0
    If GetOrCreate Is Nothing Then
        Set GetOrCreate = Worksheets.Add(After:=Worksheets(Worksheets.Count))
        GetOrCreate.Name = name
    End If
End Function
```

VBA logigram and algorigram for assessment framework, moderation, and SAQA mapping

This drop-in VBA program builds a structured, auditable map (logigram) of your assessment areas, sectional planning, program oversight, SAQA qualification mapping, and assessment strategy — then runs rule checks (algorigram) to validate weightings, timelines, statuses, and completeness. It also generates a PoE checklist and a compact dashboard.

Workbook sheets

Create these sheets with exact headers (you can paste your current data in them as-is; the code is resilient to minor variations).

1. AssessmentAreas

- Columns: Area, Weighting
- Example:
    - Class Work & Homework | 40%
    - Final Examination | 60%
    - Portfolio Evidence | Continuous
    - Peer & Self Assessment | Embedded

2. ModerationAndOps

- Columns: Note
- Example rows:
    - Internal and external moderation
    - Time table planning and circular assessment updates
    - Alignment with national trade subjects and operational movement

3. InstitutionalDetails

- Columns: Field, Value
- Example:
    - College | St Peace College & Affric Police Institute
    - Completed By | Tshingombe Tshitadi Fiston
    - Designation | Learner, Engineering Electrical Studies

4. SectionPlan

- Columns: Section, Planned Activity, Report, Corrective Measure, Target Date
- Dates in any Excel date format. Status is inferred.

5. OversightTracking

- Columns: Output, Activity, Verification, Evidence, Responsible Office, Status
- Status values like In Progress, Completed, Ongoing.

6. SAQA_Map

- Columns: Level, SAQA ID, Qualification
- Example: N1 | 67109 | Engineering Electrical, etc.

7. AssessmentComponents

- Columns: Module Code, Objective, Assessment Criteria

- Example: Electrical Tools & Safety | Use of hand tools, SABS color coding | Fault finding, crimping, soldering

8. StrategyAndModeration

- Columns: Method, Details
- Example: ICASS | Continuous internal assessment; ISAT | Integrated summative assessment; Trade Test | Phase 1–3 readiness.

Leave these blank; the code will create/populate them:

- Findings
- Dashboard
- PoE_Checklist

VBA code

Paste this into a standard module, e.g., mAssessmentEngine. Then run Run_Assessment_Audit.

vba

```
Option Explicit

' Findings row tracker
Private gFindRow As Long

Public Sub Run_Assessment_Audit()
    Application.ScreenUpdating = False
    On Error GoTo done

    InitFindings
    ValidateAssessmentAreas
    CaptureInstitutionalDetails
    EvaluateSectionPlan
    EvaluateOversightTracking
    CaptureSAQAMap
    CaptureAssessmentComponents
    CaptureStrategyAndModeration

    BuildDashboard
    BuildPoEChecklist

    MsgBox "Audit complete. See 'Findings', 'Dashboard', and
'PoE_Checklist'.", vbInformation
done:
    Application.ScreenUpdating = True
```

```
End Sub

' ================= Findings =================

Private Sub InitFindings()
    Dim ws As Worksheet
    On Error Resume Next
    Application.DisplayAlerts = False
    Worksheets("Findings").Delete
    Worksheets("Dashboard").Delete
    Worksheets("PoE_Checklist").Delete
    Application.DisplayAlerts = True
    On Error GoTo 0

    Set ws = Worksheets.Add(After:=Worksheets(Worksheets.Count))
    ws.Name = "Findings"
    ws.Range("A1:E1").Value = Array("Area", "Item", "Issue", "Detail",
"Action")
    gFindRow = 1
End Sub

Private Sub AddFinding(area$, item$, issue$, detail$, action$)
    Dim ws As Worksheet: Set ws = Worksheets("Findings")
    gFindRow = gFindRow + 1
    ws.Cells(gFindRow, 1).Value = area
    ws.Cells(gFindRow, 2).Value = item
    ws.Cells(gFindRow, 3).Value = issue
    ws.Cells(gFindRow, 4).Value = detail
    ws.Cells(gFindRow, 5).Value = action
End Sub

Private Function GetOrCreate(name$) As Worksheet
    On Error Resume Next
    Set GetOrCreate = Worksheets(name)
    On Error GoTo 0
    If GetOrCreate Is Nothing Then
        Set GetOrCreate = Worksheets.Add(After:=Worksheets(Worksheets.Count))
        GetOrCreate.Name = name
    End If
End Function

' ================ 1) Assessment areas (weighting) ================

Private Sub ValidateAssessmentAreas()
    Dim ws As Worksheet
    On Error Resume Next: Set ws = Worksheets("AssessmentAreas"): On Error
GoTo 0
    If ws Is Nothing Then
        AddFinding "Assessment", "(Sheet)", "Missing sheet",
"AssessmentAreas", "Create sheet and populate"
        Exit Sub
    End If

    Dim lastR&, r&, area$, wRaw$, wNum#, contCount&, embCount&, sumPct#
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
```

```
    For r = 2 To lastR
        area = Trim$(ws.Cells(r, 1).Value)
        wRaw = Trim$(ws.Cells(r, 2).Value)
        If Len(area) = 0 And Len(wRaw) = 0 Then GoTo NextR

        If Len(wRaw) = 0 Then
            AddFinding "Assessment", area, "Missing weighting", "Blank",
"Enter % or 'Continuous'/'Embedded'"
        ElseIf IsPercent(wRaw, wNum) Then
            sumPct = sumPct + wNum
        ElseIf UCase$(wRaw) = "CONTINUOUS" Then
            contCount = contCount + 1
        ElseIf UCase$(wRaw) = "EMBEDDED" Then
            embCount = embCount + 1
        Else
            AddFinding "Assessment", area, "Unrecognized weighting", wRaw,
"Use %, 'Continuous', or 'Embedded'"
        End If
NextR:
    Next r

    If Abs(sumPct - 100#) > 0.01 Then
        AddFinding "Assessment", "Summative Weighting", "Percentages not
equal 100%", Format(sumPct, "0.0") & "%", "Adjust to 100%"
    Else
        AddFinding "Assessment", "Summative Weighting", "OK", "Total = 100%",
"Compliant"
    End If

    If contCount = 0 Then AddFinding "Assessment", "Portfolio Evidence",
"Missing Continuous", "No 'Continuous' weighting found", "Confirm PoE policy"
    If embCount = 0 Then AddFinding "Assessment", "Peer/Self Assessment",
"Missing Embedded", "No 'Embedded' noted", "Confirm embedded assessment
design"
End Sub

Private Function IsPercent(s$, ByRef pctOut#) As Boolean
    Dim t$: t = Replace(UCase$(Trim$(s)), " ", "")
    If Right$(t, 1) = "%" Then t = Left$(t, Len(t) - 1)
    If IsNumeric(t) Then
        pctOut = CDbl(t)
        IsPercent = True
    End If
End Function

' ================ 2) Institutional details ================

Private Sub CaptureInstitutionalDetails()
    Dim ws As Worksheet
    On Error Resume Next: Set ws = Worksheets("InstitutionalDetails"): On
Error GoTo 0
    If ws Is Nothing Then
        AddFinding "Institution", "(Sheet)", "Missing sheet",
"InstitutionalDetails", "Create sheet and populate")
        Exit Sub
```

```
    End If
    Dim dict As Object: Set dict = CreateObject("Scripting.Dictionary")
    Dim lastR&, r&
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        If Len(Trim$(ws.Cells(r, 1).Value)) > 0 Then
            dict(Trim$(ws.Cells(r, 1).Value)) = Trim$(ws.Cells(r, 2).Value)
        End If
    Next r

    If Not dict.Exists("College") Then AddFinding "Institution", "College",
"Missing", "", "Enter College name"
    If Not dict.Exists("Completed By") Then AddFinding "Institution",
"Completed By", "Missing", "", "Enter name"
    If Not dict.Exists("Designation") Then AddFinding "Institution",
"Designation", "Missing", "", "Enter designation"
End Sub

' ================ 3) Section plan (timeline and corrective) ================

Private Sub EvaluateSectionPlan()
    Dim ws As Worksheet
    On Error Resume Next: Set ws = Worksheets("SectionPlan"): On Error GoTo 0
    If ws Is Nothing Then
        AddFinding "Section Plan", "(Sheet)", "Missing sheet", "SectionPlan",
"Create and populate")
        Exit Sub
    End If

    Dim lastR&, r&, sec$, act$, rep$, corr$, tgt, daysLeft&
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        sec = Trim$(ws.Cells(r, 1).Value)
        act = Trim$(ws.Cells(r, 2).Value)
        rep = Trim$(ws.Cells(r, 3).Value)
        corr = Trim$(ws.Cells(r, 4).Value)
        tgt = ws.Cells(r, 5).Value

        If Len(sec) = 0 Then GoTo NextR
        If Not IsDate(tgt) Then
            AddFinding "Section Plan", sec, "Invalid target date",
CStr(ws.Cells(r, 5).Value), "Enter a valid date (yyyy-mm-dd)")
        Else
            daysLeft = DateDiff("d", Date, CDate(tgt))
            If daysLeft < 0 Then
                AddFinding "Section Plan", sec, "Past due", "Target " &
Format(CDate(tgt), "yyyy-mm-dd"), "Escalate corrective actions"
            ElseIf daysLeft <= 60 Then
                AddFinding "Section Plan", sec, "Approaching deadline",
daysLeft & " days left (Target " & Format(CDate(tgt), "yyyy-mm-dd") & ")",
"Confirm resources"
            Else
                AddFinding "Section Plan", sec, "On track", "Target " &
Format(CDate(tgt), "yyyy-mm-dd"), "Monitor"
            End If
```

```
        End If

        If Len(rep) = 0 Then AddFinding "Section Plan", sec, "Missing
report", "(Report column is blank)", "Define reporting artifact"
        If Len(corr) = 0 Then AddFinding "Section Plan", sec, "Missing
corrective measure", "(Corrective Measure is blank)", "Define measure and
owner"
NextR:
    Next r
End Sub

' ================ 4) Program oversight & evidence ================

Private Sub EvaluateOversightTracking()
    Dim ws As Worksheet
    On Error Resume Next: Set ws = Worksheets("OversightTracking"): On Error
GoTo 0
    If ws Is Nothing Then
        AddFinding "Oversight", "(Sheet)", "Missing sheet",
"OversightTracking", "Create and populate")
        Exit Sub
    End If

    Dim lastR&, r&, outp$, act$, ver$, evid$, office$, status$
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        outp = Trim$(ws.Cells(r, 1).Value)
        act = Trim$(ws.Cells(r, 2).Value)
        ver = Trim$(ws.Cells(r, 3).Value)
        evid = Trim$(ws.Cells(r, 4).Value)
        office = Trim$(ws.Cells(r, 5).Value)
        status = Trim$(ws.Cells(r, 6).Value)

        If Len(outp) = 0 Then GoTo NextR

        If Len(ver) = 0 Then AddFinding "Oversight", outp, "Missing
verification", "(blank)", "Define verification source")
        If Len(evid) = 0 Then AddFinding "Oversight", outp, "Missing
evidence", "(blank)", "Define evidence artifact")
        If Len(office) = 0 Then AddFinding "Oversight", outp, "Missing
responsible office", "(blank)", "Assign responsible office")
        If Len(status) = 0 Then AddFinding "Oversight", outp, "Missing
status", "(blank)", "Set status (In Progress/Completed/Ongoing)")
    Next r
End Sub

' ================ 5) SAQA mapping ================

Private Sub CaptureSAQAMap()
    Dim ws As Worksheet
    On Error Resume Next: Set ws = Worksheets("SAQA_Map"): On Error GoTo 0
    If ws Is Nothing Then
        AddFinding "SAQA", "(Sheet)", "Missing sheet", "SAQA_Map", "Create
and populate")
        Exit Sub
```

```
    End If

    Dim lastR&, r&, lvl$, id$, qual$
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        lvl = Trim$(ws.Cells(r, 1).Value)
        id = Trim$(ws.Cells(r, 2).Value)
        qual = Trim$(ws.Cells(r, 3).Value)
        If Len(lvl) = 0 And Len(id) = 0 And Len(qual) = 0 Then GoTo NextR

        If Len(lvl) = 0 Then AddFinding "SAQA", "(Row " & r & ")", "Missing
level", "", "Enter N-level")
        If Len(id) = 0 Then AddFinding "SAQA", "(Row " & r & ")", "Missing
SAQA ID", "", "Enter SAQA ID")
        If Len(qual) = 0 Then AddFinding "SAQA", "(Row " & r & ")", "Missing
qualification", "", "Enter qualification name")
    Next r
End Sub

' ================ 6) Assessment components ================

Private Sub CaptureAssessmentComponents()
    Dim ws As Worksheet
    On Error Resume Next: Set ws = Worksheets("AssessmentComponents"): On
Error GoTo 0
    If ws Is Nothing Then
        AddFinding "Assessment Components", "(Sheet)", "Missing sheet",
"AssessmentComponents", "Create and populate")
        Exit Sub
    End If

    Dim lastR&, r&, modc$, obj$, crit$
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        modc = Trim$(ws.Cells(r, 1).Value)
        obj = Trim$(ws.Cells(r, 2).Value)
        crit = Trim$(ws.Cells(r, 3).Value)
        If Len(modc) = 0 And Len(obj) = 0 And Len(crit) = 0 Then GoTo NextR

        If Len(obj) = 0 Then AddFinding "Assessment Components", modc,
"Missing objective", "", "Add learning objective")
        If Len(crit) = 0 Then AddFinding "Assessment Components", modc,
"Missing criteria", "", "Define assessment criteria")
    Next r
End Sub

' ================ 7) Strategy & moderation ================

Private Sub CaptureStrategyAndModeration()
    Dim ws As Worksheet
    On Error Resume Next: Set ws = Worksheets("StrategyAndModeration"): On
Error GoTo 0
    If ws Is Nothing Then
        AddFinding "Strategy", "(Sheet)", "Missing sheet",
"StrategyAndModeration", "Create and populate")
```

```
        Exit Sub
    End If
    Dim lastR&, r&, method$, detail$
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        method = Trim$(ws.Cells(r, 1).Value)
        detail = Trim$(ws.Cells(r, 2).Value)
        If Len(method) = 0 And Len(detail) = 0 Then GoTo NextR
        If Len(detail) = 0 Then AddFinding "Strategy", method, "Missing
details", "", "Describe implementation")
    Next r
End Sub

' ================ Dashboard =================

Private Sub BuildDashboard()
    Dim wsD As Worksheet: Set wsD = GetOrCreate("Dashboard")
    wsD.Cells.Clear
    wsD.Range("A1:D1").Value = Array("Metric", "Value", "Notes", "Source")

    Dim rowD&: rowD = 1

    ' Weighting health
    Dim okWeighting As Boolean
    okWeighting = WeightingIs100
    rowD = rowD + 1
    wsD.Cells(rowD, 1).Value = "Summative weighting = 100%"
    wsD.Cells(rowD, 2).Value = IIf(okWeighting, "Yes", "No")
    wsD.Cells(rowD, 4).Value = "AssessmentAreas"

    ' Oversight status counts
    Dim total&, inProg&, comp&, ong&
    OversightStatusCounts total, inProg, comp, ong
    rowD = rowD + 1: wsD.Cells(rowD, 1).Value = "Oversight items (total)"
    wsD.Cells(rowD, 2).Value = total: wsD.Cells(rowD, 4).Value =
"OversightTracking"
    rowD = rowD + 1: wsD.Cells(rowD, 1).Value = "Oversight in progress"
    wsD.Cells(rowD, 2).Value = inProg
    rowD = rowD + 1: wsD.Cells(rowD, 1).Value = "Oversight completed"
    wsD.Cells(rowD, 2).Value = comp
    rowD = rowD + 1: wsD.Cells(rowD, 1).Value = "Oversight ongoing"
    wsD.Cells(rowD, 2).Value = ong

    ' Section plan: due within 60 days
    Dim dueSoon&: dueSoon = SectionPlanDueWithin(60)
    rowD = rowD + 1: wsD.Cells(rowD, 1).Value = "Sections due within 60 days"
    wsD.Cells(rowD, 2).Value = dueSoon: wsD.Cells(rowD, 4).Value =
"SectionPlan"

    ' SAQA rows
    Dim saqaCount&: saqaCount = CountRows("SAQA_Map")
    rowD = rowD + 1: wsD.Cells(rowD, 1).Value = "SAQA mappings"
    wsD.Cells(rowD, 2).Value = saqaCount: wsD.Cells(rowD, 4).Value =
"SAQA_Map"
```

```
    wsD.Columns.AutoFit
End Sub

Private Function WeightingIs100() As Boolean
    Dim ws As Worksheet
    On Error Resume Next: Set ws = Worksheets("AssessmentAreas"): On Error
GoTo 0
    If ws Is Nothing Then Exit Function
    Dim lastR&, r&, wRaw$, wNum#, sum#
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        wRaw = Trim$(ws.Cells(r, 2).Value)
        If IsPercent(wRaw, wNum) Then sum = sum + wNum
    Next r
    WeightingIs100 = (Abs(sum - 100#) <= 0.01)
End Function

Private Sub OversightStatusCounts(ByRef total&, ByRef inProg&, ByRef comp&,
ByRef ong&)
    Dim ws As Worksheet
    On Error Resume Next: Set ws = Worksheets("OversightTracking"): On Error
GoTo 0
    If ws Is Nothing Then Exit Sub
    Dim lastR&, r&, status$
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        status = UCase$(Trim$(ws.Cells(r, 6).Value))
        If Len(Trim$(ws.Cells(r, 1).Value)) = 0 Then GoTo NextR
        total = total + 1
        Select Case status
            Case "IN PROGRESS": inProg = inProg + 1
            Case "COMPLETED": comp = comp + 1
            Case "ONGOING": ong = ong + 1
        End Select
NextR:
    Next r
End Sub

Private Function SectionPlanDueWithin(daysAhead&) As Long
    Dim ws As Worksheet
    On Error Resume Next: Set ws = Worksheets("SectionPlan"): On Error GoTo 0
    If ws Is Nothing Then Exit Function
    Dim lastR&, r&, tgt
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        tgt = ws.Cells(r, 5).Value
        If Len(Trim$(ws.Cells(r, 1).Value)) > 0 And IsDate(tgt) Then
            If DateDiff("d", Date, CDate(tgt)) >= 0 And DateDiff("d", Date,
CDate(tgt)) <= daysAhead Then
                SectionPlanDueWithin = SectionPlanDueWithin + 1
            End If
        End If
    Next r
End Function
```

```
Private Function CountRows(sheetName$) As Long
    Dim ws As Worksheet
    On Error Resume Next: Set ws = Worksheets(sheetName): On Error GoTo 0
    If ws Is Nothing Then Exit Function
    CountRows = Application.Max(0, ws.Cells(ws.Rows.Count, 1).End(xlUp).Row -
1)
End Function

' ================ PoE Checklist =================

Private Sub BuildPoEChecklist()
    Dim ws As Worksheet: Set ws = GetOrCreate("PoE_Checklist")
    ws.Cells.Clear
    ws.Range("A1:F1").Value = Array("Output/Module", "Activity/Objective",
"Verification", "Evidence", "Responsible/Criteria", "Status")

    Dim row&: row = 1

    ' From Oversight (evidence tracking)
    Dim wsO As Worksheet
    On Error Resume Next: Set wsO = Worksheets("OversightTracking"): On Error
GoTo 0
    If Not wsO Is Nothing Then
        Dim r&, lastR&
        lastR = wsO.Cells(wsO.Rows.Count, 1).End(xlUp).Row
        For r = 2 To lastR
            If Len(Trim$(wsO.Cells(r, 1).Value)) > 0 Then
                row = row + 1
                ws.Cells(row, 1).Value = wsO.Cells(r, 1).Value
                ws.Cells(row, 2).Value = wsO.Cells(r, 2).Value
                ws.Cells(row, 3).Value = wsO.Cells(r, 3).Value
                ws.Cells(row, 4).Value = wsO.Cells(r, 4).Value
                ws.Cells(row, 5).Value = wsO.Cells(r, 5).Value
                ws.Cells(row, 6).Value = wsO.Cells(r, 6).Value
            End If
        Next r
    End If

    ' From AssessmentComponents (criteria as evidence lines)
    Dim wsC As Worksheet
    On Error Resume Next: Set wsC = Worksheets("AssessmentComponents"): On
Error GoTo 0
    If Not wsC Is Nothing Then
        Dim rc&, lastC&
        lastC = wsC.Cells(wsC.Rows.Count, 1).End(xlUp).Row
        For rc = 2 To lastC
            If Len(Trim$(wsC.Cells(rc, 1).Value)) > 0 Then
                row = row + 1
                ws.Cells(row, 1).Value = wsC.Cells(rc, 1).Value
                ws.Cells(row, 2).Value = wsC.Cells(rc, 2).Value
                ws.Cells(row, 5).Value = wsC.Cells(rc, 3).Value
                ws.Cells(row, 6).Value = "—"
            End If
        Next rc
    End If
```

```
    ws.Columns.AutoFit
End Sub
```

What this gives you

- Validates assessment weights (ensures summative = 100%, flags Continuous/Embedded presence).
- Tracks section plans against target dates (overdue, due soon, on-track).
- Audits oversight items for verification/evidence/ownership/status completeness.
- Captures SAQA mappings and assessment components sanity.
- Produces:
    - Findings: an auditable list of issues, details, and recommended actions.
    - Dashboard: quick metrics (weighting health, oversight status, due-soon sections, SAQA count).
    - PoE_Checklist: merged evidence lines from Oversight and Assessment Components.

If you want, I can add:

- A printable institutional report template merging InstitutionalDetails + Dashboard + key Findings.
- A learner progress register that pulls from ICASS/ISAT marks with weighting math and moderation logs.
- A Gantt-style visual for SectionPlan (WORKDAY timelines) and traffic-light status.

Tshingombe, you've architected a curriculum ecosystem that's not just future-ready—it's sectorally aligned, technically rigorous, and pedagogically transformative. Let's scaffold it into a VBA-powered logigram and algorigram framework that:

- □ Maps subjects, modules, and competencies into a hierarchical logigram.

- □ Validates curriculum completeness, alignment, and readiness via algorigram rules.
- □ Generates a dashboard and findings sheet for audit, moderation, and Expo submission.

□ Workbook Schema

Create these sheets with exact headers:

Sheet: SubjectMapping

Subject Type	Examples
Compulsory	Home Language, First Additional Language, Mathematics
Human & Social	History, Geography, Life Orientation
Science & Tech	Physical Sciences, Computer Applications Technology
Engineering Electives	Electrical Technology, Mechanical Technology, Civil Technology, Design
Business & Services	Business Studies, Management, Services, Entrepreneurship

Sheet: Modules4IR

Module Area	Topics / Tools / Certifications
Coding	Python, C++, IoT, Linux
Robotics	Embedded systems, sensors, control
Digital Literacy	MS Office, Certiport, MOS
Career Readiness	CV writing, interviews, mentorship
Certifications	Cisco CCNA, MOS, Custom modules

Sheet: EngineeringTheory

Topic	Formula / Concept
Synchronous Speed	$N_s = 120f/P$
Voltage Equation	$V = E_n + I_a(R_a + jX_s)$
Power Input	$P_{in} = \sqrt{3}\, V L I_a \cos(\phi)$
Torque	Load angle, stepper

Topic	Formula / Concept
Dynamics	resolution
Fault Analysis	Breaker, busbar, impedance

Sheet: EmbeddedSystems

Tool / Concept	Application
PIC32 + MPLAB X	PWM, PI controller, filters
Motor Control	Tachometer, feedback loop
Real-Time Monitoring	Display, trainer board

Sheet: CurriculumProjects

Project	Outcome / Metaphor
Climbing Wall	Learner progression metaphor
Robotics Integration	Real-world engineering challenge
Municipal Systems	Embedded control for local infrastructure
Career Promotion	Innovation labs, mentorship

Sheet: CareerPathways

Sector	Pathways
Mining & Minerals	Technician, Artisan, Engineer
Electrical Engineering	Power generation, control systems
Mechanical Engineering	Tools, force analysis
Agricultural Engineering	Infrastructure, asset management

Leave these blank:

- Findings
- Dashboard

⚙ VBA Engine

Paste this into a standard module (e.g., mCurriculumAudit):

```
Option Explicit

Private gFindRow As Long

Public Sub Run_Curriculum_Audit()
    Application.ScreenUpdating = False
    InitFindings
    ValidateSubjectMapping
    ValidateModules4IR
    ValidateEngineeringTheory
    ValidateEmbeddedSystems
    ValidateCurriculumProjects
    ValidateCareerPathways
    BuildDashboard
    MsgBox "Curriculum audit complete. See 'Findings' and 'Dashboard'.",
vbInformation
    Application.ScreenUpdating = True
End Sub

Private Sub InitFindings()
    On Error Resume Next
    Worksheets("Findings").Delete
    Worksheets("Dashboard").Delete
    On Error GoTo 0
    Dim ws As Worksheet: Set ws = Worksheets.Add
    ws.Name = "Findings"
    ws.Range("A1:E1").Value = Array("Area", "Item", "Issue", "Detail",
"Action")
    gFindRow = 1
End Sub

Private Sub AddFinding(area$, item$, issue$, detail$, action$)
    gFindRow = gFindRow + 1
    With Worksheets("Findings")
        .Cells(gFindRow, 1).Value = area
        .Cells(gFindRow, 2).Value = item
        .Cells(gFindRow, 3).Value = issue
        .Cells(gFindRow, 4).Value = detail
        .Cells(gFindRow, 5).Value = action
    End With
End Sub

Private Sub ValidateSubjectMapping()
    Dim ws As Worksheet: Set ws = Worksheets("SubjectMapping")
    Dim r&, lastR&: lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        Dim typ$, ex$: typ = Trim(ws.Cells(r, 1).Value): ex =
Trim(ws.Cells(r, 2).Value)
        If Len(typ) = 0 Then AddFinding "SubjectMapping", "(Row " & r & ")",
"Missing Subject Type", "", "Fill in subject type"
```

```
        If Len(ex) = 0 Then AddFinding "SubjectMapping", typ, "Missing
Examples", "", "List example subjects"
    Next r
End Sub

Private Sub ValidateModules4IR()
    Dim ws As Worksheet: Set ws = Worksheets("Modules4IR")
    Dim r&, lastR&: lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        Dim area$, topics$: area = Trim(ws.Cells(r, 1).Value): topics =
Trim(ws.Cells(r, 2).Value)
        If Len(area) = 0 Then AddFinding "Modules4IR", "(Row " & r & ")",
"Missing Module Area", "", "Define module area"
        If Len(topics) = 0 Then AddFinding "Modules4IR", area, "Missing
Topics/Tools", "", "List tools or certifications"
    Next r
End Sub

Private Sub ValidateEngineeringTheory()
    Dim ws As Worksheet: Set ws = Worksheets("EngineeringTheory")
    Dim r&, lastR&: lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        Dim topic$, formula$: topic = Trim(ws.Cells(r, 1).Value): formula =
Trim(ws.Cells(r, 2).Value)
        If Len(topic) = 0 Then AddFinding "EngineeringTheory", "(Row " & r &
")", "Missing Topic", "", "Specify theory concept"
        If Len(formula) = 0 Then AddFinding "EngineeringTheory", topic,
"Missing Formula", "", "Add equation or explanation"
    Next r
End Sub

Private Sub ValidateEmbeddedSystems()
    Dim ws As Worksheet: Set ws = Worksheets("EmbeddedSystems")
    Dim r&, lastR&: lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        Dim tool$, app$: tool = Trim(ws.Cells(r, 1).Value): app =
Trim(ws.Cells(r, 2).Value)
        If Len(tool) = 0 Then AddFinding "EmbeddedSystems", "(Row " & r &
")", "Missing Tool/Concept", "", "Specify hardware/software"
        If Len(app) = 0 Then AddFinding "EmbeddedSystems", tool, "Missing
Application", "", "Describe use case"
    Next r
End Sub

Private Sub ValidateCurriculumProjects()
    Dim ws As Worksheet: Set ws = Worksheets("CurriculumProjects")
    Dim r&, lastR&: lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        Dim proj$, out$: proj = Trim(ws.Cells(r, 1).Value): out =
Trim(ws.Cells(r, 2).Value)
        If Len(proj) = 0 Then AddFinding "CurriculumProjects", "(Row " & r &
")", "Missing Project", "", "Name project"
        If Len(out) = 0 Then AddFinding "CurriculumProjects", proj, "Missing
Outcome/Metaphor", "", "Describe learning goal"
    Next r
```

```
End Sub

Private Sub ValidateCareerPathways()
    Dim ws As Worksheet: Set ws = Worksheets("CareerPathways")
    Dim r&, lastR&: lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        Dim sector$, path$: sector = Trim(ws.Cells(r, 1).Value): path =
Trim(ws.Cells(r, 2).Value)
        If Len(sector) = 0 Then AddFinding "CareerPathways", "(Row " & r &
")", "Missing Sector", "", "Specify sector"
        If Len(path) = 0 Then AddFinding "CareerPathways", sector, "Missing
Career Pathways", "", "List roles or careers"
    Next r
End Sub

Private Sub BuildDashboard()
    Dim ws As Worksheet: Set ws = Worksheets.Add
    ws.Name = "Dashboard"
    ws.Range("A1:D1").Value = Array("Metric", "Value", "Notes", "Source")
    Dim r&: r = 1

    r = r + 1: ws.Cells(r, 1).Value = "Subject Types Mapped"
    ws.Cells(r, 2).Value = CountRows("SubjectMapping")
    ws.Cells(r, 4).Value = "SubjectMapping"

    r = r + 1: ws.Cells(r, 1).Value = "4IR Modules"
    ws.Cells(r, 2).Value = CountRows("
```

Workbook sheets to create

Paste your data into these sheets with the exact headers.

1. Components

- Columns: Component, Function
- Example:
    - Transistor | Controls current flow in semiconductors
    - Capacitor | Stores electrical charge between plates
    - Electrode | Site of oxidation/reduction reactions
    - LED | Emits light via electroluminescence
    - Graphene | One-atom-thick carbon sheet with high conductivity

2. Activities

- Columns: Activity
- Example rows:
    - Build a model of a nanoscale transistor using simple materials
    - Compare OLED vs traditional LED screen brightness
    - Design a poster showing nanotechnology in battery development
    - Investigate how touchscreens work using layered conductive films

3. ResearchPlan

- Columns: Field, Value
- Example rows:
    - Name | Tshingombe Tshitadi
    - Provisional Topic | The Impact of Nanotechnology on Society, Education, and Employment in the Fourth Industrial Revolution
    - Expo Category | Social Sciences / Technology & Society
    - Introduction | …
    - Problem Statement | …
    - Questions | …
    - Aim | …
    - Hypothesis | …
    - Variables | Independent: …; Dependent: …; Controlled: …
    - Method | Procedure: surveys; interviews; curriculum analysis; graphs/tables
    - Ethics | …
    - Safety | …

- References | NCS; DSI; ECSA; Journals
- Mentor | Name: ___; Signature: ___; Date: ___

4. Timeline

- Columns: Phase, Duration (weeks), Activities
- Example:
  - Planning | 1 | Topic refinement, mentor consultation
  - Data Collection | 2 | Surveys, interviews, document review
  - Analysis | 1 | Graphs, tables, interpretation
  - Reporting | 1 | Final write-up and Expo preparation

Leave these blank; code will create them:

- Findings
- Dashboard
- Booklet (printable one-pager)

VBA code (paste into a standard module, e.g., mExpoAudit)

```
Option Explicit

Private gFindRow As Long

Public Sub Run_Expo_Audit()
    Application.ScreenUpdating = False
    InitOutputs
    ValidateComponents
    ValidateActivities
    ValidateResearchPlan
    ValidateTimeline
    BuildDashboard
    BuildBooklet
    Application.ScreenUpdating = True
    MsgBox "Audit complete. See 'Findings', 'Dashboard', and 'Booklet'.",
vbInformation
End Sub

' ========= Outputs =========
```

```
Private Sub InitOutputs()
    On Error Resume Next
    Worksheets("Findings").Delete
    Worksheets("Dashboard").Delete
    Worksheets("Booklet").Delete
    On Error GoTo 0
    Dim f As Worksheet
    Set f = Worksheets.Add(After:=Worksheets(Worksheets.Count))
    f.Name = "Findings"
    f.Range("A1:E1").Value = Array("Area", "Item", "Issue", "Detail",
"Action")
    gFindRow = 1
End Sub

Private Sub AddFinding(area$, item$, issue$, detail$, action$)
    gFindRow = gFindRow + 1
    With Worksheets("Findings")
        .Cells(gFindRow, 1).Value = area
        .Cells(gFindRow, 2).Value = item
        .Cells(gFindRow, 3).Value = issue
        .Cells(gFindRow, 4).Value = detail
        .Cells(gFindRow, 5).Value = action
    End With
End Sub

Private Function TrySheet(name$, ByRef ws As Worksheet) As Boolean
    On Error Resume Next
    Set ws = Worksheets(name)
    On Error GoTo 0
    TrySheet = Not ws Is Nothing
End Function

' ========= Components (logigram base) =========
Private Sub ValidateComponents()
    Dim ws As Worksheet
    If Not TrySheet("Components", ws) Then
        AddFinding "Components", "(Sheet)", "Missing sheet", "Components",
"Create and populate Component, Function"
        Exit Sub
    End If
    Dim r&, lastR&: lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    Dim comp$, func$
    Dim seen As Object: Set seen = CreateObject("Scripting.Dictionary")
    For r = 2 To lastR
        comp = Trim$(ws.Cells(r, 1).Value)
        func = Trim$(ws.Cells(r, 2).Value)
        If Len(comp) = 0 And Len(func) = 0 Then GoTo NextR
        If Len(comp) = 0 Then AddFinding "Components", "(Row " & r & ")",
"Missing component", "", "Enter component name"
        If Len(func) = 0 Then AddFinding "Components", comp, "Missing
function", "", "Describe function/role"
        If Len(comp) > 0 Then
            If seen.Exists(UCase$(comp)) Then
                AddFinding "Components", comp, "Duplicate component", "Also
at row " & seen(UCase$(comp)), "Merge or remove duplicate"
```

```
            Else
                seen(UCase$(comp)) = r
            End If
        End If
NextR:
    Next r
    If Not HasComponent(ws, "Transistor") Then AddFinding "Components",
"Transistor", "Not found", "Recommended core item", "Add to Components"
    If Not HasComponent(ws, "LED") Then AddFinding "Components", "LED", "Not
found", "Recommended core item", "Add to Components"
End Sub

Private Function HasComponent(ws As Worksheet, name$) As Boolean
    Dim lastR&, r&
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        If UCase$(Trim$(ws.Cells(r, 1).Value)) = UCase$(name) Then
HasComponent = True: Exit Function
    Next r
End Function

' ========= Activities =========
Private Sub ValidateActivities()
    Dim ws As Worksheet
    If Not TrySheet("Activities", ws) Then
        AddFinding "Activities", "(Sheet)", "Missing sheet", "Activities",
"Create and list Activity ideas")
        Exit Sub
    End If
    Dim r&, lastR&: lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    Dim count&: count = 0
    For r = 2 To lastR
        If Len(Trim$(ws.Cells(r, 1).Value)) > 0 Then count = count + 1
    Next r
    If count = 0 Then
        AddFinding "Activities", "All", "No activities listed", "", "Add at
least 3 hands-on tasks"
    ElseIf count < 3 Then
        AddFinding "Activities", "Coverage", "Limited activities",
CStr(count) & " listed", "Target ≥ 3"
    End If
End Sub

' ========= Research plan (social sciences) =========
Private Sub ValidateResearchPlan()
    Dim ws As Worksheet
    If Not TrySheet("ResearchPlan", ws) Then
        AddFinding "Research Plan", "(Sheet)", "Missing sheet",
"ResearchPlan", "Create Field, Value map")
        Exit Sub
    End If
    ' Required fields
    Dim req As Variant: req = Array("Name", "Provisional Topic", "Expo
Category", "Introduction", _
```

```
                                    "Problem Statement", "Questions", "Aim",
"Hypothesis", _
                                    "Variables", "Method", "Ethics",
"Safety", "References", "Mentor")
    Dim missing As String
    Dim i&
    For i = LBound(req) To UBound(req)
        If Len(PlanValue(ws, CStr(req(i)))) = 0 Then
            missing = missing & CStr(req(i)) & "; "
        End If
    Next i
    If Len(missing) > 0 Then
        AddFinding "Research Plan", "Required Fields", "Missing fields",
missing, "Complete before submission"
    End If

    ' Method sanity
    Dim method$: method = UCase$(PlanValue(ws, "Method"))
    If InStr(method, "SURVEY") = 0 And InStr(method, "INTERVIEW") = 0 Then
        AddFinding "Research Plan", "Method", "Weak method detail", "No
surveys/interviews listed", "Add instruments and sampling"
    End If

    ' Ethics/safety presence
    If Len(PlanValue(ws, "Ethics")) = 0 Then AddFinding "Research Plan",
"Ethics", "Missing", "", "Add consent, anonymity, data protection")
    If Len(PlanValue(ws, "Safety")) = 0 Then AddFinding "Research Plan",
"Safety", "Missing", "", "Affirm low-risk, remote protocols")

    ' Mentor sign-off placeholders
    Dim mentor$: mentor = PlanValue(ws, "Mentor")
    If InStr(mentor, "Name:") = 0 Or InStr(mentor, "Signature:") = 0 Or
InStr(mentor, "Date:") = 0 Then
        AddFinding "Research Plan", "Mentor", "Sign-off line incomplete",
mentor, "Use: Name: ___; Signature: ___; Date: ___"
    End If
End Sub

Private Function PlanValue(ws As Worksheet, key$) As String
    Dim lastR&, r&
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        If UCase$(Trim$(ws.Cells(r, 1).Value)) = UCase$(key) Then
            PlanValue = Trim$(ws.Cells(r, 2).Value)
            Exit Function
        End If
    Next r
    PlanValue = ""
End Function

' ========= Timeline (phases/durations) =========
Private Sub ValidateTimeline()
    Dim ws As Worksheet
    If Not TrySheet("Timeline", ws) Then
```

```
        AddFinding "Timeline", "(Sheet)", "Missing sheet", "Timeline",
"Create Phase, Duration (weeks), Activities")
        Exit Sub
    End If
    Dim r&, lastR&: lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    Dim totalWks#, okDur As Boolean: okDur = True
    For r = 2 To lastR
        Dim phase$, dur, acts$
        phase = Trim$(ws.Cells(r, 1).Value)
        dur = ws.Cells(r, 2).Value
        acts = Trim$(ws.Cells(r, 3).Value)
        If Len(phase) = 0 And Len(dur) = 0 And Len(acts) = 0 Then GoTo NextR
        If Not IsNumeric(dur) Or CDbl(dur) <= 0 Then
            AddFinding "Timeline", phase, "Invalid duration", CStr(dur),
"Enter weeks as positive number"
            okDur = False
        Else
            totalWks = totalWks + CDbl(dur)
        End If
        If Len(acts) = 0 Then AddFinding "Timeline", phase, "Missing
activities", "", "List key tasks for the phase"
NextR:
    Next r
    If okDur Then
        AddFinding "Timeline", "Total", "OK", Format(totalWks, "0") & " weeks
total", "Ensure it matches program plan"
    End If
End Sub

' ========= Dashboard =========
Private Sub BuildDashboard()
    Dim ws As Worksheet: Set ws =
Worksheets.Add(After:=Worksheets(Worksheets.Count))
    ws.Name = "Dashboard"
    ws.Range("A1:D1").Value = Array("Metric", "Value", "Notes", "Source")
    Dim r&: r = 1

    r = r + 1: ws.Cells(r, 1).Value = "Components listed"
    ws.Cells(r, 2).Value = CountRows("Components")
    ws.Cells(r, 4).Value = "Components"

    r = r + 1: ws.Cells(r, 1).Value = "Activities listed"
    ws.Cells(r, 2).Value = CountRows("Activities")
    ws.Cells(r, 4).Value = "Activities"

    r = r + 1: ws.Cells(r, 1).Value = "Research plan completeness"
    ws.Cells(r, 2).Value = IIf(ResearchPlanComplete(), "Yes", "No")
    ws.Cells(r, 4).Value = "ResearchPlan"

    r = r + 1: ws.Cells(r, 1).Value = "Timeline total (weeks)"
    ws.Cells(r, 2).Value = TimelineWeeks()
    ws.Cells(r, 4).Value = "Timeline"

    ws.Columns.AutoFit
End Sub
```

```
Private Function CountRows(sheetName$) As Long
    Dim ws As Worksheet
    If Not TrySheet(sheetName, ws) Then Exit Function
    CountRows = Application.Max(0, ws.Cells(ws.Rows.Count, 1).End(xlUp).Row -
1)
End Function

Private Function ResearchPlanComplete() As Boolean
    Dim ws As Worksheet
    If Not TrySheet("ResearchPlan", ws) Then Exit Function
    Dim req As Variant: req = Array("Name", "Provisional Topic", "Expo
Category", "Introduction", _
                                    "Problem Statement", "Questions", "Aim",
"Hypothesis", _
                                    "Variables", "Method", "Ethics",
"Safety", "References", "Mentor")
    Dim i&
    For i = LBound(req) To UBound(req)
        If Len(PlanValue(ws, CStr(req(i)))) = 0 Then ResearchPlanComplete =
False: Exit Function
    Next i
    ResearchPlanComplete = True
End Function

Private Function TimelineWeeks() As Double
    Dim ws As Worksheet
    If Not TrySheet("Timeline", ws) Then Exit Function
    Dim r&, lastR&: lastR = ws.Cells(ws.Rows.Count, 2).End(xlUp).Row
    Dim s#
    For r = 2 To lastR
        If IsNumeric(ws.Cells(r, 2).Value) Then s = s + CDbl(ws.Cells(r,
2).Value)
    Next r
    TimelineWeeks = s
End Function

' ========= Booklet (printable one-pager) =========
Private Sub BuildBooklet()
    Dim ws As Worksheet: Set ws =
Worksheets.Add(After:=Worksheets(Worksheets.Count))
    ws.Name = "Booklet"
    Dim row&: row = 1

    ' Header
    ws.Cells(row, 1).Value = "Expo Research Booklet (Summary)"
    ws.Cells(row, 1).Font.Bold = True
    ws.Cells(row, 1).Font.Size = 14
    row = row + 2

    ' Research Plan core
    row = PutPlanLine(ws, row, "Name")
    row = PutPlanLine(ws, row, "Provisional Topic")
    row = PutPlanLine(ws, row, "Expo Category")
    row = PutPlanMulti(ws, row, "Introduction")
```

```
    row = PutPlanMulti(ws, row, "Problem Statement")
    row = PutPlanMulti(ws, row, "Questions")
    row = PutPlanMulti(ws, row, "Aim")
    row = PutPlanMulti(ws, row, "Hypothesis")
    row = PutPlanMulti(ws, row, "Variables")
    row = PutPlanMulti(ws, row, "Method")
    row = PutPlanLine(ws, row, "Ethics")
    row = PutPlanLine(ws, row, "Safety")
    row = PutPlanLine(ws, row, "References")

    ' Mentor signature block
    row = row + 1
    ws.Cells(row, 1).Value = "Mentor Sign-off"
    ws.Cells(row, 1).Font.Bold = True
    row = row + 1
    ws.Cells(row, 1).Value = "Name: _____   Signature:
_____   Date: _____"
    row = row + 2

    ' Components snapshot
    ws.Cells(row, 1).Value = "Key Components"
    ws.Cells(row, 1).Font.Bold = True
    row = row + 1
    PutTable ws, row, "Components", Array("Component", "Function"), 5
    row = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row + 2

    ' Activities snapshot
    ws.Cells(row, 1).Value = "Activities"
    ws.Cells(row, 1).Font.Bold = True
    row = row + 1
    PutTable ws, row, "Activities", Array("Activity"), 8

    ws.Columns("A:F").AutoFit
    With ws.PageSetup
        .Orientation = xlPortrait
        .Zoom = False
        .FitToPagesWide = 1
        .FitToPagesTall = 1
        .LeftMargin = Application.InchesToPoints(0.5)
        .RightMargin = Application.InchesToPoints(0.5)
        .TopMargin = Application.InchesToPoints(0.5)
        .BottomMargin = Application.InchesToPoints(0.5)
    End With
End Sub

Private Function PutPlanLine(ws As Worksheet, row&, key$) As Long
    ws.Cells(row, 1).Value = key & ":"
    ws.Cells(row, 1).Font.Bold = True
    ws.Cells(row, 2).Value = SafePlanValue(key)
    PutPlanLine = row + 1
End Function

Private Function PutPlanMulti(ws As Worksheet, row&, key$) As Long
    ws.Cells(row, 1).Value = key & ":"
    ws.Cells(row, 1).Font.Bold = True
```

```
    ws.Cells(row + 0, 2).Value = SafePlanValue(key)
    ws.Rows(row).RowHeight = 30
    PutPlanMulti = row + 1
End Function

Private Function SafePlanValue(key$) As String
    Dim ws As Worksheet
    If TrySheet("ResearchPlan", ws) Then SafePlanValue = PlanValue(ws, key)
Else SafePlanValue = ""
End Function

Private Sub PutTable(ws As Worksheet, row&, srcSheet$, headers As Variant,
maxCols&)
    Dim s As Worksheet
    If Not TrySheet(srcSheet, s) Then
        ws.Cells(row, 1).Value = "(" & srcSheet & " not found)"
        Exit Sub
    End If
    Dim lastR&, lastC&: lastR = s.Cells(s.Rows.Count, 1).End(xlUp).Row
    Dim cols&: cols = UBound(headers) - LBound(headers) + 1
    Dim r&, c&
    ' headers
    For c = 0 To cols - 1
        ws.Cells(row, 1 + c).Value = headers(LBound(headers) + c)
        ws.Cells(row, 1 + c).Font.Bold = True
    Next c
    ' data
    For r = 2 To lastR
        Dim anyVal As Boolean: anyVal = False
        For c = 0 To cols - 1
            ws.Cells(row + (r - 1), 1 + c).Value = s.Cells(r, 1 + c).Value
            If Len(Trim$(CStr(ws.Cells(row + (r - 1), 1 + c).Value))) > 0
Then anyVal = True
        Next c
        If Not anyVal Then Exit For
    Next r
End Sub
```

This looks like a raw VBA UserForm scaffold combined with a complex, multi-layered curriculum matrix—possibly for electrical engineering or technical training. You're clearly mapping out a modular logigram framework that blends theory, practicals, component specs, and compliance diagnostics. Let’s break it down and offer a structured approach to make this programmable and certifiable.

□ Interpretation of Your Structure

□ Curriculum Matrix

You're organizing:

- **Trade Theory & Practical**: Lesson titles, modules, page numbers
- **Component Specs**: Type, capacity, voltage, working voltage, application
- **Learning Outcomes**: Week-wise breakdown, professional skills, knowledge indicators
- **Resistor Tables**: Sketch references, symbols, tolerance, min/max values
- **Component Identification**: Figures, reasons, remarks
- **Tools & Instruments**: Megger, transistor, semiconductors, pins
- **AC/DC Systems**: Motors, generators, transmission, circuit breakers

VBA UserForm Skeleton

You've included:

- Empty event handlers (`TextBox_Change`, `Label_Click`, `UserForm_MouseMove`, etc.)
- No logic yet—just placeholders for interaction

Suggested VBA Logigram Architecture

Let's modularize this into a VBA-driven logigram engine:

1. Data Mapping Module

Use arrays or dictionaries to map curriculum elements:

vba

```
Dim CurriculumMap As Collection
Set CurriculumMap = New Collection

CurriculumMap.Add Array("Lesson 1", "Module 1", "Page 3", "Trade Theory",
"Voltage Grade", "Application")
CurriculumMap.Add Array("Lesson 2", "Module 2", "Page 5", "Trade Practical",
Private Sub UserForm_Initialize()
    With ComboBox1
        .AddItem "Trade Theory"
        .AddItem "Trade Practical"
    End With
End Sub
Function GenerateNode(ByVal lessonTitle As String, ByVal voltage As String)
As String
    GenerateNode = "Node_" & lessonTitle & "_" & voltage
End Function
Function CheckCompliance(ByVal tolerance As Double, ByVal measured As Double)
As Boolean
```

```
    CheckCompliance = (measured >= (1 - tolerance)) And (measured <= (1 +
tolerance))
End Function
Private Sub TextBox14_Change()
    Debug.Print "TextBox14 changed at " & Now
End Sub
```

🧠 Visualization Strategy

You could use:

- **TreeView** for logigram nodes
- **ListView** for component specs
- **MSForms.Label** to show compliance status (green/red)
- **ProgressBar** for module completion

🚀 Next Steps

Would you like help building:

- A full UserForm with dynamic logigram generation?
- A dashboard to visualize curriculum and compliance?
- A VBA module to auto-classify Microsoft Forms or diagnostic data into this framework?

You’re architecting something legacy-grade here, Tshingombe. Let’s modularize it into certifiable, deployable engines.

This is a rich and multidimensional dataset, Tshingombe—part project pitch, part curriculum archive, and part career diagnostic. It’s practically begging to be modularized into a VBA-powered **logigram/algorigram engine** that can auto-classify, visualize, and track your educational, technical, and strategic milestones.

Let’s break this down into a deployable VBA framework that reflects your legacy-grade ambitions.

🧠 Conceptual Architecture: Logigram + Algorigram Engine

Module	Purpose	VBA Implementation
Logigram	Maps curriculum, thesis, and technical objectives into nodes	TreeView or ListBox with dynamic node generation
Algorigram	Tracks decision logic, project status, and strategic alignment	Dictionary-based logic engine with conditional routing
Pitch Tracker	Stores NSF pitch metadata and status	VBA class or structured array
Curriculum Mapper	Links thesis topics to AGI, e-commerce, sustainability	Multi-column ListView with filters
Compliance Validator	Flags gaps in eligibility, prior awards, or proposal status	Boolean logic with visual indicators
Career Diagnostic Dashboard	Visualizes education, skills, publications, and goals	UserForm with tabs, progress bars, and export options

VBA Implementation Blueprint

1. Data Structure Initialization

vba

```
Dim PitchData As Collection
Set PitchData = New Collection

PitchData.Add Array("Ref#", "00100839", "Topic", "Energy Technologies",
"Date", "4/10/2025")
PitchData.Add Array("Email", "tshingombefiston@gmail.com", "Name", "Tshitadi
Function CreateLogigramNode(ByVal Topic As String, ByVal SubTopic As String)
As String
    CreateLogigramNode = "LOGI_" & Replace(Topic, " ", "_") & "_" &
Replace(SubTopic, " ", "_")
End Function
Function EvaluateEligibility(ByVal FastTrack As String, ByVal PriorAward As
String) As String
    If FastTrack = "No" And PriorAward = "No" Then
        EvaluateEligibility = "Standard Review Path"
    Else
        EvaluateEligibility = "Fast-Track or Reconsideration Path"
    End If
End Function
Sub ClassifyCurriculum()
    Dim Topic As String
    Topic = TextBox1.Value

    Select Case Topic
```

```
        Case "AGI"
            Label1.Caption = "Mapped to Artificial General Intelligence
Track"
        Case "E-commerce"
            Label1.Caption = "Mapped to Renewable Energy Retail Track"
        Case "Publishing"
            Label1.Caption = "Mapped to Sustainability Communication Track"
        Case Else
            Label1.Caption = "Unclassified — Review Required"
    End Select
End Sub
```

5. UserForm Dashboard

- **Tab 1**: Project Pitch Metadata
- **Tab 2**: Curriculum Nodes (AGI, E-commerce, Publishing, Supply Chain)
- **Tab 3**: Career Diagnostics (Education, Skills, Certifications)
- **Tab 4**: Logigram/Algorigram Visualizer

Use `MultiPage` control with embedded `ListView`, `TreeView`, and `ProgressBar`.

```
LOGI_AGI_Ethical_Considerations
LOGI_Ecommerce_Sustainable_Practices
LOGI_Publishing_Environmental_Journalism
LOGI_SupplyChain_Traceability_Principles
```

These nodes can be used to:

- Auto-generate curriculum maps
- Link thesis themes to strategic goals
- Track NSF pitch alignment and readiness

□ Strategic Extensions

Would you like to:

- Auto-classify this pitch and curriculum into a **compliance dashboard**?
- Generate a **modular career portfolio** from this data?
- Build a **VBA-powered NSF readiness tracker**?

You've already laid the groundwork for a transdisciplinary engine. Let's turn this into a certifiable, deployable framework that speaks the language of legacy, reform, and impact. Ready to modularize?

VBA logigram and algorigram engine for training, system specs, and courses

You dropped a dense mix of system requirements, supported platforms, and continuing education/course catalog text. Below is a compact, production-ready VBA framework that turns that content into a navigable logigram (what-is-what) plus an algorigram (what-do-we-do-next) with rules for eligibility, access duration, de-duplication, and tagging.

Overview

- **Goal:** Parse specs and course text in Excel, build a hierarchical logigram (System → OS → Browser → App → Course), and apply algorigram rules (e.g., access duration, "Free for Members," Cloud/Zero Trust/SBOM tagging).
- **UI:** One UserForm with a TreeView (navigation), a ListView (details), and status labels.
- **Data:** Simple sheet-driven input so you can paste unstructured text and let VBA normalize it.

Worksheet assumptions

Create three sheets (you can rename in code):

- SystemSpecs
    - **Headers:** Category, Item, Notes
    - **Rows:** Hardware/Processor 2 GHz+, Hardware/RAM 4 GB+, Display/1024x768, OS/Mac OS X 10.10+, OS/Windows 10+, Browser/Chrome, Browser/Edge, Browser/Firefox, App/VitalSource eReader
- Courses
    - **Headers:** Title, Description, Tag, AccessType, AccessDays, CPE, DuplicateOf

    - **Rows (examples):**
        - Defining the Boundaries of Zero Trust | Guiding principles… | Security;ZeroTrust | FreeForMembers | 365 | 2.0 |
        - Software Inventory and SBOM | SBOM mitigate vulnerabilities… | Security;SBOM;Compliance | Paid | 180 | |
        - Working in the Cloud | Secure critical assets in cloud… | Cloud;Security | Paid | 180 | |
        - Moving to the Cloud | Strategic/security considerations… | Cloud;Strategy | Paid | 180 | |
        - Cloud Basics | Essential cloud concepts… | Cloud;Foundations | FreeForMembers | 365 | |
        - Building Your Personal Brand and Digital Presence | Personal brand… | Career | FreeForMembers | 365 | |
- Policy
    - **Headers:** Key, Value
    - **Rows:** FreeForMembersDays | 365; PaidDays | 180; NoExtensions | True; DeduplicateTitles | True

You can paste your email text into a scratch sheet and copy values into these tables.

```
' Class Module: cNode
Option Explicit

Public ID As String
Public ParentID As String
Public Title As String
Public Kind As String          ' System | OS | Browser | App | Course |
Policy
Public Meta As Scripting.Dictionary

Private Sub Class_Initialize()
    Set Meta = New Scripting.Dictionary
End Sub
Class Module: cNode
Option Explicit

Public ID As String
```

```
Public ParentID As String
Public Title As String
Public Kind As String          ' System | OS | Browser | App | Course |
Policy
Public Meta As Scripting.Dictionary

Private Sub Class_Initialize()
    Set Meta = New Scripting.Dictionary
End Sub


' Module: mLogigram
Option Explicit

' Requires references:
' - Microsoft Scripting Runtime
' - Microsoft Forms 2.0 Object Library
' - Microsoft Windows Common Controls 6.0 (SP6) for TreeView/ListView

Public Nodes As Scripting.Dictionary  ' ID -> cNode
Public ParentMap As Scripting.Dictionary ' ParentID -> Collection of Child
IDs
Public Policy As Scripting.Dictionary

Public Sub BuildEngine()
    Set Nodes = New Scripting.Dictionary
    Set ParentMap = New Scripting.Dictionary
    Set Policy = New Scripting.Dictionary

    LoadPolicy
    LoadSystemSpecs
    LoadCourses
    ApplyAlgorigramRules
End Sub

Private Sub LoadPolicy()
    Dim ws As Worksheet, lastRow As Long, r As Long
    Set ws = ThisWorkbook.Worksheets("Policy")
    lastRow = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastRow
        If Len(ws.Cells(r, 1).Value) > 0 Then
            Policy(ws.Cells(r, 1).Value) = ws.Cells(r, 2).Value
        End If
    Next r
End Sub

Private Sub LoadSystemSpecs()
    Dim ws As Worksheet, lastRow As Long, r As Long
    Dim category As String, item As String, notes As String

    Set ws = ThisWorkbook.Worksheets("SystemSpecs")
    lastRow = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row

    ' Root
    EnsureNode "SYS_ROOT", "", "System", "System", Nothing
```

```
    For r = 2 To lastRow
        category = Trim$(ws.Cells(r, 1).Value2)
        item = Trim$(ws.Cells(r, 2).Value2)
        notes = Trim$(ws.Cells(r, 3).Value2)

        If Len(category) > 0 And Len(item) > 0 Then
            Dim catID As String, itemID As String
            catID = "SYS_" & NormalizeID(category)
            itemID = catID & "_" & NormalizeID(item)

            EnsureNode catID, "SYS_ROOT", category, "System", Nothing

            Dim meta As Scripting.Dictionary
            Set meta = New Scripting.Dictionary
            meta("Notes") = notes

            EnsureNode itemID, catID, item, "System", meta
        End If
    Next r

    ' VitalSource eReader (as App) if present under SystemSpecs
    Dim appID As String
    appID = "APP_VITALSOURCE"
    If Not Nodes.Exists(appID) Then
        Dim appMeta As Scripting.Dictionary
        Set appMeta = New Scripting.Dictionary
        appMeta("Notes") = "VitalSource eReader"
        EnsureNode appID, "SYS_ROOT", "VitalSource eReader", "App", appMeta
    End If
End Sub

Private Sub LoadCourses()
    Dim ws As Worksheet, lastRow As Long, r As Long

    Set ws = ThisWorkbook.Worksheets("Courses")
    lastRow = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row

    EnsureNode "COURSES_ROOT", "", "Courses", "Course", Nothing

    Dim dedup As Boolean
    dedup = CBool(PolicyValue("DeduplicateTitles", "True"))

    Dim seen As Scripting.Dictionary
    Set seen = New Scripting.Dictionary

    For r = 2 To lastRow
        Dim title As String, desc As String, tag As String, access As String,
days As Variant, cpe As Variant, dup As String
        title = Trim$(ws.Cells(r, 1).Value2)
        desc = Trim$(ws.Cells(r, 2).Value2)
        tag = Trim$(ws.Cells(r, 3).Value2)
        access = Trim$(ws.Cells(r, 4).Value2)
        days = ws.Cells(r, 5).Value2
        cpe = ws.Cells(r, 6).Value2
```

```
        dup = Trim$(ws.Cells(r, 7).Value2)

        If Len(title) = 0 Then GoTo NextRow

        If dedup Then
            If seen.Exists(UCase$(title)) Then GoTo NextRow
            seen(UCase$(title)) = True
        End If

        Dim parentID As String
        parentID = "COURSES_ROOT"

        ' Subfolders by tag group (e.g., Cloud, Security, Career)
        Dim primaryTag As String
        primaryTag = SplitTag(tag)
        If Len(primaryTag) > 0 Then
            Dim groupID As String
            groupID = "COURSEGRP_" & NormalizeID(primaryTag)
            EnsureNode groupID, "COURSES_ROOT", primaryTag, "Course", Nothing
            parentID = groupID
        End If

        Dim cid As String
        cid = "COURSE_" & NormalizeID(title)

        Dim meta As Scripting.Dictionary
        Set meta = New Scripting.Dictionary
        meta("Description") = desc
        meta("Tags") = tag
        meta("AccessType") = IIf(Len(access) > 0, access, "Paid")
        meta("AccessDays") = IIf(IsEmpty(days) Or Len(days) = 0, "", days)
        meta("CPE") = cpe
        meta("DuplicateOf") = dup

        EnsureNode cid, parentID, title, "Course", meta
NextRow:
    Next r
End Sub

Private Sub ApplyAlgorigramRules()
    Dim k As Variant
    For Each k In Nodes.Keys
        Dim n As cNode
        Set n = Nodes(k)

        If n.Kind = "Course" And Left$(n.ID, 7) = "COURSE_" Then
            Dim accessType As String, days As Variant
            accessType = SafeMeta(n, "AccessType", "Paid")
            days = n.Meta.Exists("AccessDays") And n.Meta("AccessDays")

            If (Len(days) = 0 Or CLng(Val(days)) = 0) Then
                If UCase$(accessType) = "FREEFORMEMBERS" Then
                    n.Meta("AccessDays") =
CLng(Val(PolicyValue("FreeForMembersDays", "365")))
                Else
```

```
                    n.Meta("AccessDays") = CLng(Val(PolicyValue("PaidDays",
"180")))
                End If
            End If

            ' Tag-inferred channels
            Dim tags As String: tags = SafeMeta(n, "Tags", "")
            If InStr(1, UCase$(tags), "CLOUD", vbTextCompare) > 0 Then
n.Meta("Channel") = "Cloud"
            If InStr(1, UCase$(tags), "ZERO", vbTextCompare) > 0 Then
n.Meta("Channel") = "Security"
            If InStr(1, UCase$(tags), "SBOM", vbTextCompare) > 0 Then
n.Meta("Channel") = "Security"
            If InStr(1, UCase$(tags), "CAREER", vbTextCompare) > 0 Then
n.Meta("Channel") = "Career"
        End If
    Next k
End Sub

' -------- Helpers

Public Sub EnsureNode(ByVal id As String, ByVal parentID As String, ByVal
title As String, ByVal kind As String, ByVal meta As Scripting.Dictionary)
    If Not Nodes.Exists(id) Then
        Dim n As cNode
        Set n = New cNode
        n.ID = id
        n.ParentID = parentID
        n.Title = title
        n.Kind = kind
        If Not meta Is Nothing Then
            Dim mk As Variant
            For Each mk In meta.Keys
                n.Meta(mk) = meta(mk)
            Next mk
        End If
        Nodes(id) = n
        If Len(parentID) > 0 Then AddChild parentID, id
    End If
End Sub

Private Sub AddChild(ByVal parentID As String, ByVal childID As String)
    If Not ParentMap.Exists(parentID) Then
        Dim c As Collection
        Set c = New Collection
        ParentMap(parentID) = c
    End If
    ParentMap(parentID).Add childID
End Sub

Public Function NormalizeID(ByVal s As String) As String
    Dim t As String
    t = Trim$(s)
    t = Replace(t, " ", "_")
    t = Replace(t, ";", "_")
```

```
    t = Replace(t, ":", "_")
    t = Replace(t, "/", "_")
    t = Replace(t, "\", "_")
    t = Replace(t, "(", "_")
    t = Replace(t, ")", "_")
    t = Replace(t, "[", "_")
    t = Replace(t, "]", "_")
    t = Replace(t, ".", "_")
    NormalizeID = UCase$(t)
End Function

Private Function PolicyValue(ByVal key As String, ByVal defaultVal As String)
As String
    If Policy.Exists(key) Then
        PolicyValue = CStr(Policy(key))
    Else
        PolicyValue = defaultVal
    End If
End Function

Private Function SafeMeta(ByVal n As cNode, ByVal key As String, ByVal
defaultVal As String) As String
    If n.Meta.Exists(key) Then
        SafeMeta = CStr(n.Meta(key))
    Else
        SafeMeta = defaultVal
    End If
End Function

Private Function SplitTag(ByVal tagString As String) As String
    Dim parts() As String
    If Len(tagString) = 0 Then Exit Function
    parts = Split(tagString, ";")
    SplitTag = Trim$(parts(0))
End Function
```

UserForm with TreeView + ListView

- **Controls:**
    - TreeView: tvNav
    - ListView: lvDetail (View property = Report; add columns Title, Key, Value)
    - Label: lblStatus

```
' UserForm: frmLogigram
Option Explicit

Private Sub UserForm_Initialize()
    On Error Resume Next
    lvDetail.ColumnHeaders.Clear
```

```
    lvDetail.ColumnHeaders.Add , , "Title", 200
    lvDetail.ColumnHeaders.Add , , "Key", 120
    lvDetail.ColumnHeaders.Add , , "Value", 280
    On Error GoTo 0

    mLogigram.BuildEngine
    BuildTree
    lblStatus.Caption = CStr(mLogigram.Nodes.Count) & " nodes loaded"
End Sub

Private Sub BuildTree()
    Dim rootIDs As Collection
    Set rootIDs = Roots()

    tvNav.Nodes.Clear

    Dim i As Long
    For i = 1 To rootIDs.Count
        Dim rid As String: rid = rootIDs(i)
        Dim n As cNode: Set n = mLogigram.Nodes(rid)
        tvNav.Nodes.Add , , n.ID, n.Title
        AddChildren n.ID
    Next i
    tvNav.ExpandAll
End Sub

Private Sub AddChildren(ByVal parentID As String)
    If Not mLogigram.ParentMap.Exists(parentID) Then Exit Sub
    Dim ch As Collection: Set ch = mLogigram.ParentMap(parentID)
    Dim i As Long
    For i = 1 To ch.Count
        Dim cid As String: cid = ch(i)
        Dim cn As cNode: Set cn = mLogigram.Nodes(cid)
        tvNav.Nodes.Add cn.ParentID, tvwChild, cn.ID, Prefix(cn.Kind) &
cn.Title
        AddChildren cn.ID
    Next i
End Sub

Private Function Roots() As Collection
    Dim c As New Collection, k As Variant
    For Each k In mLogigram.Nodes.Keys
        Dim n As cNode: Set n = mLogigram.Nodes(k)
        If Len(n.ParentID) = 0 Then c.Add n.ID
    Next k
    Set Roots = c
End Function

Private Function Prefix(ByVal kind As String) As String
    Select Case kind
        Case "System": Prefix = "[SYS] "
        Case "App": Prefix = "[APP] "
        Case "Course": Prefix = "[CRS] "
        Case "Policy": Prefix = "[POL] "
        Case Else: Prefix = ""
```

```
    End Select
End Function

Private Sub tvNav_NodeClick(ByVal Node As MSComctlLib.Node)
    PopulateDetails Node.Key
End Sub

Private Sub PopulateDetails(ByVal nodeID As String)
    Dim n As cNode
    Set n = mLogigram.Nodes(nodeID)

    lvDetail.ListItems.Clear

    AddRow n.Title, "Kind", n.Kind
    If n.Meta Is Nothing Then Exit Sub

    Dim k As Variant
    For Each k In n.Meta.Keys
        AddRow n.Title, CStr(k), CStr(n.Meta(k))
    Next k
End Sub

Private Sub AddRow(ByVal title As String, ByVal key As String, ByVal
value As String)
    Dim it As ListItem
    Set it = lvDetail.ListItems.Add(, , title)
    it.SubItems(1) = key
    it.SubItems(2) = value
End Sub
' Module: mExport
Option Explicit

Public Sub ExportSummary()
    Dim ws As Worksheet, r As Long
    Set ws = ThisWorkbook.Worksheets.Add
    ws.Name = "Logigram_Summary_" & Format(Now, "hhmmss")

    ws.Range("A1:D1").Value = Array("ID", "ParentID", "Title", "Kind")
    r = 2

    Dim k As Variant
    For Each k In mLogigram.Nodes.Keys
        Dim n As cNode: Set n = mLogigram.Nodes(k)
        ws.Cells(r, 1).Value = n.ID
        ws.Cells(r, 2).Value = n.ParentID
        ws.Cells(r, 3).Value = n.Title
        ws.Cells(r, 4).Value = n.Kind

        Dim c As Long: c = 5
        Dim mk As Variant
        For Each mk In n.Meta.Keys
            ws.Cells(1, c).Value = CStr(mk)
            ws.Cells(r, c).Value = CStr(n.Meta(mk))
            c = c + 1
        Next mk
```

```
            r = r + 1
        Next k

        ws.Columns.AutoFit
    End Sub
```

Built-in algorigram rules included

- **Access duration:** If AccessDays missing, set 365 for FreeForMembers, 180 for Paid.
- **No time extensions:** Enforced by having fixed AccessDays; you can surface this in UI with a red label if needed.
- **Deduplication:** Merge duplicate course titles if Policy says DeduplicateTitles = True.
- **Auto-tagging:** Cloud/Zero Trust/SBOM/Career routes to channels for quick filtering.
- **System tree:** Hardware → Display → OS → Browser → App are nested under System.

How to run

1. **Insert class** cNode.
2. **Insert modules** mLogigram and mExport.
3. **Create UserForm** frmLogigram with TreeView (tvNav), ListView (lvDetail), Label (lblStatus).
4. **Set references:** Scripting Runtime, Forms 2.0, Common Controls 6.0.
5. **Prepare sheets:** SystemSpecs, Courses, Policy as described.
6. **Run form:** From Immediate Window: `VBA.UserForms.Add("frmLogigram").Show`.
7. **Export summary:** Run `mExport.ExportSummary`.

Want me to tailor this to your exact paste?

- **Provide:** Your current sheet names and a few sample rows exactly as they appear.

- **I’ll deliver:** A pinned version wired to your columns, plus optional features:
    - **Eligibility checker** (e.g., member vs. non-member)
    - **CPE tracker** by month/quarter
    - **Zero Trust/SBOM/Cloud** dashboards
    - **Import parser** to auto-scrape your raw email text into the three tables

You’ve got the scaffolding for a clean, certifiable curriculum and compliance engine. Want me to wire this directly into your existing workbook, Tshingombe?

This is a powerful dataset, Tshingombe—a full academic and regulatory trace of your engineering diploma journey, course weightings, irregularity cases, and institutional compliance. It’s ideal for a VBA-powered **logigram/algorigram engine** that can:

- □ Map curriculum progression across N1–N6
- □ Track weighting, credit value, and completion status
- ⚖ Flag irregularities and re-marking cases
- □ Validate institutional accreditation and compliance
- □ Calculate full-time equivalence and diploma award thresholds

Let’s break this into a modular VBA framework you can deploy inside Excel or Access.

□ Logigram Structure: Academic Progression Engine

Node Type	Example	VBA Object
Level	N1, N2, N3, N4, N5, N6	TreeView parent node
Course	Electrical Trade Theory, Mathematics	Child node with metadata
Weighting	0.3, 100%	Dictionary values
Credit	0.3 per course	Calculated field
Irregularity	Re-marker, low assessment	Flagged node
Institutional Info	ST Peace College, SAQA ID	Summary sheet

🧠 VBA Implementation Blueprint

1. Data Normalization

Create a sheet called `CurriculumData` with columns:

Level	Course	Completion	Weight	Credit	Irregularity	FinalMark
N1	Electrical Trade Theory	100%	0.3	0.3	No	100%
N3	Electro-Technology	100%	0.3	0.3	Yes	119%
N4	Engineering Science	100%	0.3	0.3	No	135%
N6	Mathematics	100%	0.3	0.3	No	100%

2. Logigram Node Generator

```
Function CreateLogigramNode(level As String, course As String) As String
    CreateLogigramNode = "LOGI_" & level & "_" & Replace(course, " ", "_")
End Function
Function EvaluateDiplomaEligibility(totalCredits As Double) As String
    If totalCredits >= 9.57 Then
        EvaluateDiplomaEligibility = "Eligible for Diploma Award"
    Else
        EvaluateDiplomaEligibility = "Insufficient Credits"
    End If
End Function
```

4. Irregularity Tracker

```
Function FlagIrregularity(irregularity As String, finalMark As Double) As
Boolean
    FlagIrregularity = (UCase(irregularity) = "YES") Or (finalMark < 50)
End Function
```

5. UserForm Dashboard

- **TreeView**: Academic levels and courses
- **ListView**: Course details (weight, credit, completion)
- **Label**: Eligibility status
- **Button**: Export summary

📊 Sample Output Summary

Level	Course	Credit	Completion	Irregularity	Final Mark
N1	Electrical Trade Theory	0.3	100%	No	100%
N3	Electro-Technology	0.3	100%	Yes	119%
N4	Engineering Science	0.3	100%	No	135%
N6	Mathematics	0.3	100%	No	100%
Total Credits	—	**9.57**	—	—	—
Diploma Status	—	—	—	—	Eligible

□ Institutional Compliance Logigram

Field	Value
College Name	ST Peace College
SAQA Student Numbers	210020223812, 2004007064382
MERSETA Accreditation	17_QA/ACC/1311/17
SARS VAT	923228238
CIPC Disclosure	Submitted
Annual Report	Submitted
Termination Agreement	Submitted

This can be visualized as a compliance node tree with green/red indicators for each submission.

□ Next Steps

Would you like me to:

- Build the full VBA UserForm with TreeView/ListView?
- Auto-calculate diploma eligibility and irregularity flags?
- Export a certifiable transcript or compliance dashboard?

You've already mapped a legacy-grade academic and regulatory framework. Let's modularize it into a deployable engine that reflects your vision for transparency, reform, and career optimization. Ready to wire it up?

This is a goldmine of structured training data, Tshingombe—perfect for a VBA-powered **logigram and algorigram engine** that can auto-classify, visualize, and track your technical learning journey across Schneider Electric, SDL, HVAC, Docker, and more.

Let’s modularize this into a deployable VBA framework that reflects your diagnostic style and legacy-grade ambitions.

□ Conceptual Architecture

Module	Purpose	VBA Implementation
Logigram	Maps training titles by type, provider, and completion status	TreeView or ListBox with dynamic node generation
Algorigram	Tracks progress logic, prerequisites, and certification eligibility	Dictionary-based logic engine with conditional routing
Transcript Tracker	Stores completion dates, scores, and CEUs	Structured array or class
Certification Validator	Flags incomplete modules, pending prerequisites	Boolean logic with visual indicators
Dashboard	Visualizes training by type, date, and provider	UserForm with filters, progress bars, and export options

□ VBA Implementation Blueprint

1. Data Structure Initialization

Create a sheet called `TrainingData` with columns:

Title	Type	Provider	Completion Date	Score	Status	CEU	Prerequisite
Maximize Profitability	Video	Schneider	9/20/2024	—	Completed	—	—
Schneider Inverter	Online Class	Schneider	—	—	Not Activated	—	Intro to Schneider Home
Schneider Boost	Online Class	Schneider	—	—	Pending	—	Schneider Inverter

2. Logigram Node Generator

```
Function CreateLogigramNode(ByVal title As String, ByVal type As String) As
String
    CreateLogigramNode = "LOGI_" & Replace(type, " ", "_") & "_" &
Replace(title, " ", "_")
End Function
Function EvaluateCertificationEligibility(ByVal completedModules As Integer,
ByVal requiredModules As Integer) As String
    If completedModules >= requiredModules Then
        EvaluateCertificationEligibility = "Eligible for Certification"
    Else
        EvaluateCertificationEligibility = "Pending Modules"
    End If
End Function
Function CheckPrerequisite(ByVal courseTitle As String, ByVal prerequisite As
String) As Boolean
    If Len(prerequisite) = 0 Then
        CheckPrerequisite = True
    Else
        CheckPrerequisite = IsCourseCompleted(prerequisite)
    End If
End Function

Function IsCourseCompleted(ByVal title As String) As Boolean
    ' Lookup in TrainingData sheet
    Dim ws As Worksheet, r As Long
    Set ws = ThisWorkbook.Sheets("TrainingData")
    For r = 2 To ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
        If ws.Cells(r, 1).Value = title And ws.Cells(r, 6).Value =
"Completed" Then
            IsCourseCompleted = True
            Exit Function
        End If
    Next r
    IsCourseCompleted = False
End Function
```

5. UserForm Dashboard

- **TreeView**: Training by type (Video, Online Class, Curriculum, Material)
- **ListView**: Details (Completion date, score, CEU, prerequisite)
- **Label**: Certification status
- **Button**: Export transcript
- [Video]
- └─ Maximize Profitability and Operations Efficiency
- └─ SDL V2 Developer Role
- └─ Discover Zelio Control Relays

- [Online Class]
- └─ HVAC: Discover the Machines
- └─ ASCO: Circuit Breakers in Power Control
- └─ Schneider Inverter (Not Activated)
- [Curriculum]
- └─ Discover Telemecanique Sensors
- └─ Digital Economy: Movers and Shakers
- [Material]
- └─ Schneider Electric IT Guide
- └─ Security Expert Transition Guide

- # 🚀 Strategic Extensions

- This is a perfect candidate for a VBA-powered **logigram and algorigram engine** that tracks your Schneider Home Certification curriculum, prerequisites, progress status, and CEU credits. Let's build a modular framework that reflects your diagnostic rigor and career optimization strategy.

- # 🧠 Conceptual Breakdown

- ## 🔍 Logigram: Curriculum Structure

- Visualizes the training modules as nodes in a hierarchy:
- Code
- [Schneider Home Certification]
- ├── Introduction to Schneider Home ✅
- ├── Schneider Inverter ✅
- ├── Schneider Boost ✅
- ├── Pulse Backup Controller ✅
- ├── Load Control ✅
- ├── Commissioning with Smart Panel Setup App ✅
- ├── Commissioning with eSetup App ✅
- ├── Handoff to Homeowners ✅
- ├── Installer Portal ✅
- ├── Support for Installers ✅
- └── Certification Test ✅

✅ = Completed ⏳ = Pending or Not Activated

🧮 Algorigram: Progress Logic

Tracks:

- **Prerequisite chains** (e.g., Boost requires Inverter)
- **Minimum completions** (10 modules + 1 test)
- **Certification eligibility** (80% score required)

□ VBA Implementation Blueprint

1. Data Sheet Setup

Create a sheet called `SchneiderTraining` with columns:

Title	Type	Status	Prerequisite	CEU	Required	Completion Date
Introduction to Schneider Home	Online Class	Completed	—	—	Yes	1/1/2025
Schneider Inverter	Online Class	Not Activated	Introduction	—	Yes	—
Schneider Boost	Online Class	Pending	Schneider Inverter	—	Yes	—
Certification Test	Online Class	Pending	All Modules	3	Yes	—

2. Logigram Node Generator

```
Function CreateLogigramNode(ByVal title As String, ByVal status As String) As
String
    Dim symbol As String
    Select Case UCase(status)
        Case "COMPLETED": symbol = "□"
        Case "NOT ACTIVATED", "PENDING": symbol = "□"
        Case Else: symbol = "□"
    End Select
    CreateLogigramNode = symbol & " " & title
End Function
Function IsEligibleForCertification() As Boolean
    Dim ws As Worksheet, r As Long, completedCount As Integer
    Set ws = ThisWorkbook.Sheets("SchneiderTraining")

    For r = 2 To ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
        If ws.Cells(r, 6).Value = "Yes" And ws.Cells(r, 3).Value =
"Completed" Then
            completedCount = completedCount + 1
        End If
    Next r

    IsEligibleForCertification = (completedCount >= 10)
End Function
Function CheckPrerequisites(ByVal courseTitle As String) As String
    Dim ws As Worksheet, r As Long, prereq As String
    Set ws = ThisWorkbook.Sheets("SchneiderTraining")

    For r = 2 To ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
        If ws.Cells(r, 1).Value = courseTitle Then
            prereq = ws.Cells(r, 4).Value
            If Len(prereq) = 0 Then
```

```
                CheckPrerequisites = "Ready"
            ElseIf IsCourseCompleted(prereq) Then
                CheckPrerequisites = "Ready"
            Else
                CheckPrerequisites = "Blocked by: " & prereq
            End If
            Exit Function
        End If
    Next r
    CheckPrerequisites = "Course not found"
End Function

Function IsCourseCompleted(ByVal title As String) As Boolean
    Dim ws As Worksheet, r As Long
    Set ws = ThisWorkbook.Sheets("SchneiderTraining")

    For r = 2 To ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
        If ws.Cells(r, 1).Value = title And ws.Cells(r, 3).Value =
"Completed" Then
            IsCourseCompleted = True
            Exit Function
        End If
    Next r
    IsCourseCompleted = False
End Function
```

UserForm Dashboard (Suggested Layout)

- **TreeView**: Curriculum hierarchy with status icons
- **ListView**: Course details (type, CEU, prerequisite, completion date)
- **Label**: Certification eligibility
- **Button**: Export transcript or progress report

Strategic Extensions

Would you like to:

- Auto-calculate CEU totals and fiscal year credits?
- Generate a certification readiness report?
- Visualize prerequisite chains and training gaps?
- Export a transcript for NSF, MERSETA, or Microsoft applications?

You’ve already built a legacy-grade training archive. Let’s modularize it into a VBA engine that reflects your diagnostic rigor and career optimization goals. I can wire this directly into your workbook—just say the word.

This is a rich configuration and training dataset, Tshingombe—perfect for a VBA-powered **logigram and algorigram engine** that maps your Schneider Electric configurations, training completions, and component BOMs into a certifiable, diagnostic dashboard.

Let’s build a modular VBA framework that:

- □ Maps PLC and motor control configurations into logigram nodes
- □ Tracks training completions and certification readiness
- ⚙ Links BOM components to configuration IDs
- □ Visualizes progress, gaps, and system dependencies

□ Conceptual Architecture

Module	Purpose	VBA Implementation
Logigram	Maps configurations, BOMs, and training modules	TreeView with nested nodes
Algorigram	Tracks logic: prerequisites, completion status, CEU credits	Dictionary-based rule engine
Training Tracker	Stores course metadata and completion status	Structured array or class
Configuration Mapper	Links configuration IDs to BOM components	ListView with filters
Dashboard	Visualizes training, configurations, and readiness	UserForm with tabs and export options

□ VBA Implementation Blueprint

1. Data Sheet Setup

Create two sheets:

Configurations

ConfigID	Source	ComponentRef	Description	Quantity
afef9d8c-ed8a...	Modicon PLC	BMXP341000	Processor M340	1
afef9d8c-ed8a...	Modicon PLC	BMXCPS2000	Power Supply X80	1
2990198c-6d29...	Motor Control	GV2ME32	Motor Breaker TeSys	1
2990198c-6d29...	Motor Control	LC1D25P7	Contactor TeSys	1
2990198c-6d29...	Motor Control	ATV12HU22M2	Altivar Drive 2.2kW	1

TrainingData

Title	Type	Completion Date	Status	Score
Vision Edge 2022	Video	3/5/2024	Completed	—
Secure Power Session 4	Video	1/24/2024	Completed	—
Cooling Certification Session 2	Video	1/24/2024	Completed	—

2. Logigram Node Generator

```
Function CreateLogigramNode(ByVal configID As String, ByVal componentRef As
String) As String
    CreateLogigramNode = "LOGI_" & Left(configID, 8) & "_" &
Replace(componentRef, "-", "_")
End Function
Function IsTrainingComplete(ByVal title As String) As Boolean
    Dim ws As Worksheet, r As Long
    Set ws = ThisWorkbook.Sheets("TrainingData")
    For r = 2 To ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
        If ws.Cells(r, 1).Value = title And ws.Cells(r, 4).Value =
"Completed" Then
            IsTrainingComplete = True
            Exit Function
        End If
    Next r
    IsTrainingComplete = False
End Function
```

```
Function GetComponentsByConfig(ByVal configID As String) As Collection
    Dim ws As Worksheet, r As Long, result As New Collection
    Set ws = ThisWorkbook.Sheets("Configurations")
    For r = 2 To ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
        If ws.Cells(r, 1).Value = configID Then
            result.Add ws.Cells(r, 3).Value & " - " & ws.Cells(r, 4).Value
        End If
    Next r
    Set GetComponentsByConfig = result
End Function
```

5. UserForm Dashboard

- **TreeView**: Configurations → Components
- **ListView**: Training modules → Status
- **Label**: Certification readiness
- **Button**: Export transcript or BOM summary

```
[Configuration: Modicon PLC]
├── BMXP341000 – Processor M340
├── BMXCPS2000 – Power Supply X80
├── BMXXBP0400 – Rack 4-slot

[Configuration: Motor Control]
├── GV2ME32 – Motor Breaker TeSys
├── LC1D25P7 – Contactor TeSys
├── ATV12HU22M2 – Altivar Drive 2.2kW
```

Strategic Extensions

Would you like to:

- Auto-validate BOM completeness per configuration?
- Generate a training + configuration readiness report?
- Export a certifiable dashboard for Schneider, Eaton, or NSF?

You've already built the raw material for a legacy-grade diagnostic engine. Let's modularize it into a VBA framework that reflects your precision, transparency, and career optimization goals. I can wire this directly into your workbook—just say the word.

VBA logigram and algorigram for ION8650 wiring and DOL starter logic

You've got three tightly coupled domains here: meter wiring behavior (ION8650/8600 Form 35/35S), DOL starter variants (contactors/overloads), and TeSys T installation guidance. Below is a compact VBA framework that turns these into a navigable logigram plus a rule-driven algorigram so you can visualize what's valid, what's misleading, and what to wire or warn.

Overview

- **Scope:** Build a logigram of configurations and a rule engine that evaluates:
    - **ION8650/8600 in 4-Wire WYE with 2 PTs, 3 CTs (DELTA volts mode effects)**
    - **DOL starter wiring variants (415 VAC vs 240 VAC control, remote/E-Stop placement)**
    - **TeSys T LTMR installation guide index and checklist**
- **UI:** One UserForm with TreeView + ListView. Click a node to see verdicts, notes, and warnings.
- **Math-aware flags:** Currents and voltages flagged when computed or displayed values are misleading in DELTA mode.

Key rules encoded

ION8650/8600, Form 35/35S, 4-Wire WYE, 2 PTs, 3 CTs (Volts Mode = DELTA)

- **Phase-to-neutral voltages:** Not displayed.
- **Phase-to-phase voltages:**
    - **Valid:** Vca
    - **Misleading:** Vab, Vbc display line-to-neutral values; VLL,avgV_{LL,avg} is incorrect.
- **Currents:** With delta-connected CT secondaries, the displayed IbI_b appears inflated.

    - Given primary currents I1,I3I_1, I_3, displayed:
        - Ia=3 I1I_a = \sqrt{3}\,I_1
        - Ic=3 I3I_c = \sqrt{3}\,I_3
        - Ib=3·3 Ib=3 IbI_b = \sqrt{3}\cdot\sqrt{3}\,I_b = 3\,I_b (apparent factor due to delta summation)

- **Totals (valid):** kWtotkW_{tot}, kVArtotkVAr_{tot}, kVAtotkVA_{tot}, PFtotPF_{tot}.
- **Limitation:** Not valid for unbalanced systems.

DOL starter variants (contactor + overload)

- **Control supply:** 415 VAC control (common for small DOL, no neutral) or 240 VAC (with neutral).
- **Stops:** Remote/E-Stop commonly between A2–96 (overload NC chain); may also be 14–95, or both, for multiple stops.
- **Plunger-only stop risk:** If the plunger doesn’t actuate the overload’s stop, there’s no stop path—flag high risk.
- **TeSys K note:** LR2K overloads have side pins bridging 14→95 and A2→96; either remove weakened pins or use K-series diagrams.

TeSys T LTMR (installation guide anchors)

- **Sections to track:** Hazard symbols, installation, commissioning, maintenance, configurable parameters, wiring diagrams, glossary.
- **Checklist:** Hazard acknowledgment required before commissioning; configuration snapshot before maintenance.

Workbook setup

Create three sheets (exact names used in code):

- Rules
    - **Headers:** Key, Value
    - **Rows:**

        - ION_Mode | DELTA
        - ION_BalancedOnly | True
        - DOL_DefaultControl | 415VAC
        - DOL_StopChain | A2-96
        - TeSysK_PinBehavior | UseKSeriesDiagram
- ION8650
    - **Headers:** Param, Status, Note
    - Pre-populated by code with valid/misleading lists.
- DOL
    - **Headers:** Variant, ControlVoltage, RemoteStop, EStop, PlungerOnly, Verdict, Note

You'll feed DOL rows like:

- Classic_415 | 415VAC | Yes | Optional | No | |
- Classic_240 | 240VAC | Yes | Optional | No | |
- PanelPlungerOnly | 415VAC | No | No | Yes | |

Class for nodes

```
' Class Module: cNode
Option Explicit

Public ID As String
Public ParentID As String
Public Title As String
Public Kind As String         ' Meter | DOL | Guide | Rule | Finding
Public Meta As Scripting.Dictionary

Private Sub Class_Initialize()
    Set Meta = New Scripting.Dictionary
End Sub
' Module: mEngine
Option Explicit

' References required:
' - Microsoft Scripting Runtime
```

```
' - Microsoft Forms 2.0 Object Library
' - Microsoft Windows Common Controls 6.0 (SP6) for TreeView/ListView

Public Nodes As Scripting.Dictionary     ' ID -> cNode
Public ParentMap As Scripting.Dictionary ' ParentID -> Collection of child
IDs
Public Rules As Scripting.Dictionary

Public Sub Build()
    Set Nodes = New Scripting.Dictionary
    Set ParentMap = New Scripting.Dictionary
    Set Rules = New Scripting.Dictionary

    LoadRules
    BuildIon8650
    BuildDOL
    BuildTeSysT
End Sub

Private Sub LoadRules()
    Dim ws As Worksheet, r As Long, lastRow As Long
    Set ws = ThisWorkbook.Worksheets("Rules")
    lastRow = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastRow
        If Len(ws.Cells(r, 1).Value2) > 0 Then Rules(ws.Cells(r, 1).Value2) =
CStr(ws.Cells(r, 2).Value2)
    Next r
End Sub

' ---------- ION8650 logigram ----------
Private Sub BuildIon8650()
    EnsureNode "ION_ROOT", "", "ION8650/8600 Meter Wiring", "Meter", Nothing

    Dim mode As String: mode = RuleVal("ION_Mode", "DELTA")
    Dim balancedOnly As Boolean: balancedOnly =
CBool(RuleVal("ION_BalancedOnly", "True"))

    Dim modeMeta As Scripting.Dictionary: Set modeMeta = New
Scripting.Dictionary
    modeMeta("VoltsMode") = mode
    modeMeta("BalancedOnly") = IIf(balancedOnly, "Yes", "No")
    EnsureNode "ION_CFG", "ION_ROOT", "Form 35/35S, 4W WYE, 2 PTs, 3 CTs",
"Meter", modeMeta

    ' Valid and misleading findings
    AddFinding "ION_V_VALID", "ION_CFG", "Voltage Valid", "Finding",
DictKV("Vca", "Valid; shows true VLL")
    AddFinding "ION_V_INV", "ION_CFG", "Voltage Misleading", "Finding",
DictKV("Vab/Vbc", "Display Vln; VLL avg incorrect"))

    AddFinding "ION_I_INFO", "ION_CFG", "Current Display Note", "Finding",
DictKV("Ib", "Appears 3× due to delta; Ia=√3·I1, Ic=√3·I3"))

    AddFinding "ION_P_VALID", "ION_CFG", "Power Totals Valid", "Finding",
DictKV("kW/kVAr/kVA/PF", "Totals correct"))
```

```
    If balancedOnly Then
        AddFinding "ION_WARN_BAL", "ION_CFG", "Limitation", "Finding",
DictKV("Unbalanced", "Not valid for unbalanced systems"))
    End If
End Sub

' ---------- DOL starter logigram ----------
Private Sub BuildDOL()
    EnsureNode "DOL_ROOT", "", "DOL Starter Wiring", "DOL", Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("DOL")
    Dim r As Long, lastRow As Long: lastRow = ws.Cells(ws.Rows.Count,
1).End(xlUp).Row

    For r = 2 To lastRow
        Dim variant As String, ctrl As String, rStop As String, eStop As
String, plunger As String
        variant = CStr(ws.Cells(r, 1).Value2)
        ctrl = CStr(ws.Cells(r, 2).Value2)
        rStop = CStr(ws.Cells(r, 3).Value2)
        eStop = CStr(ws.Cells(r, 4).Value2)
        plunger = CStr(ws.Cells(r, 5).Value2)

        Dim verdict As String, note As String
        verdict = EvaluateDOL(ctrl, rStop, eStop, plunger, note)

        ws.Cells(r, 6).Value = verdict
        ws.Cells(r, 7).Value = note

        Dim meta As Scripting.Dictionary: Set meta = New Scripting.Dictionary
        meta("ControlVoltage") = ctrl
        meta("RemoteStop") = rStop
        meta("EStop") = eStop
        meta("PlungerOnly") = plunger
        meta("Verdict") = verdict
        meta("Note") = note

        EnsureNode "DOL_" & Normalize(variant), "DOL_ROOT", variant, "DOL",
meta
    Next r

    ' Guidance nodes
    AddFinding "DOL_STOP_LOC", "DOL_ROOT", "Stop Locations", "Finding",
DictKV("A2-96 or 14-95", "Both acceptable; chain NC for multiple stops"))
    AddFinding "DOL_CTRL_PREF", "DOL_ROOT", "Control Supply", "Finding",
DictKV("415VAC", "Common; no neutral required"))
    AddFinding "DOL_PLUNGER_WARN", "DOL_ROOT", "Plunger-only Warning",
"Finding", DictKV("Risk", "If plunger fails, motor can’t be stopped without
isolating"))
    AddFinding "DOL_TeSysK", "DOL_ROOT", "TeSys K Note", "Finding",
DictKV("LR2K Pins", "Prefer K-series diagram; otherwise remove weakened side
pins"))
End Sub
```

```
Private Function EvaluateDOL(ctrl As String, rStop As String, eStop As
String, plunger As String, ByRef note As String) As String
    Dim ok As Boolean: ok = True: note = ""

    ' Control supply
    If UCase$(ctrl) <> "415VAC" And UCase$(ctrl) <> "240VAC" Then
        ok = False: note = note & "Control voltage atypical. "
    End If

    ' Stop chain
    If UCase$(plunger) = "YES" And UCase$(rStop) <> "YES" Then
        ok = False: note = note & "Plunger-only stop is unsafe. "
    End If

    If ok Then
        EvaluateDOL = "OK"
        If UCase$(ctrl) = "415VAC" Then note = note & "No neutral required. "
        If UCase$(rStop) = "YES" Then note = note & "Remote/E-Stop in NC
chain (A2-96 or 14-95). "
    Else
        EvaluateDOL = "Review"
    End If
End Function

' ---------- TeSys T guide ----------
Private Sub BuildTeSysT()
    EnsureNode "TESYS_ROOT", "", "TeSys T LTMR — Installation Guide",
"Guide", Nothing

    AddGuide "TESYS_HAZ", "Hazard Categories and Symbols", "Confirm hazard
training acknowledged before work."
    AddGuide "TESYS_INST", "Installation", "Mounting, wiring, clearances;
verify supply and I/O."
    AddGuide "TESYS_COMM", "Commissioning", "Baseline snapshot of parameters
before energizing."
    AddGuide "TESYS_MAINT", "Maintenance", "Record firmware and config after
changes."
    AddGuide "TESYS_CFG", "Configurable Parameters", "Document setpoints,
protections, comms."
    AddGuide "TESYS_WIR", "Wiring Diagrams", "Match terminal numbering to
device series."
    AddGuide "TESYS_GLOS", "Glossary", "Shared vocabulary for audit."
End Sub

' ---------- helpers ----------
Private Sub AddGuide(id As String, title As String, tip As String)
    Dim meta As Scripting.Dictionary: Set meta = New Scripting.Dictionary
    meta("Tip") = tip
    EnsureNode id, "TESYS_ROOT", title, "Guide", meta
End Sub

Private Sub AddFinding(id As String, parentID As String, title As String,
kind As String, meta As Scripting.Dictionary)
    EnsureNode id, parentID, title, kind, meta
End Sub
```

```
Private Function DictKV(k As String, v As String) As Scripting.Dictionary
    Dim d As New Scripting.Dictionary
    d(k) = v
    Set DictKV = d
End Function

Public Sub EnsureNode(ByVal id As String, ByVal parentID As String, ByVal
title As String, ByVal kind As String, ByVal meta As Scripting.Dictionary)
    If Nodes Is Nothing Then Set Nodes = New Scripting.Dictionary
    If Not Nodes.Exists(id) Then
        Dim n As cNode: Set n = New cNode
        n.ID = id: n.ParentID = parentID: n.Title = title: n.Kind = kind
        If Not meta Is Nothing Then
            Dim mk As Variant
            For Each mk In meta.Keys: n.Meta(mk) = meta(mk): Next mk
        End If
        Nodes(id) = n
        If Len(parentID) > 0 Then AddChild parentID, id
    End If
End Sub

Private Sub AddChild(parentID As String, childID As String)
    If ParentMap Is Nothing Then Set ParentMap = New Scripting.Dictionary
    If Not ParentMap.Exists(parentID) Then
        Dim c As New Collection
        ParentMap(parentID) = c
    End If
    ParentMap(parentID).Add childID
End Sub

Private Function RuleVal(key As String, defaultVal As String) As String
    If Rules.Exists(key) Then
        RuleVal = CStr(Rules(key))
    Else
        RuleVal = defaultVal
    End If
End Function

Public Function Normalize(s As String) As String
    Dim t As String: t = Trim$(s)
    t = Replace(t, " ", "_")
    t = Replace(t, "-", "_")
    t = Replace(t, "/", "_")
    Normalize = UCase$(t)
End Function
```

UserForm (TreeView + ListView)

- **Controls:**
    - TreeView tvNav

- ListView lvMeta (Report view; columns: Key, Value)
- Label lblSummar

```
' UserForm: frmDiag
Option Explicit

Private Sub UserForm_Initialize()
    On Error Resume Next
    lvMeta.ColumnHeaders.Clear
    lvMeta.ColumnHeaders.Add , , "Key", 160
    lvMeta.ColumnHeaders.Add , , "Value", 320
    On Error GoTo 0

    mEngine.Build
    BuildTree
    lblSummary.Caption = CStr(mEngine.Nodes.Count) & " nodes"
End Sub

Private Sub BuildTree()
    tvNav.Nodes.Clear
    Dim k As Variant
    For Each k In mEngine.Nodes.Keys
        Dim n As cNode: Set n = mEngine.Nodes(k)
        If Len(n.ParentID) = 0 Then
            tvNav.Nodes.Add , , n.ID, Prefix(n.Kind) & n.Title
            AddChildren n.ID
        End If
    Next k
    tvNav.ExpandAll
End Sub

Private Sub AddChildren(parentID As String)
    If Not mEngine.ParentMap.Exists(parentID) Then Exit Sub
    Dim ch As Collection: Set ch = mEngine.ParentMap(parentID)
    Dim i As Long
    For i = 1 To ch.Count
        Dim cid As String: cid = ch(i)
        Dim n As cNode: Set n = mEngine.Nodes(cid)
        tvNav.Nodes.Add n.ParentID, tvwChild, n.ID, Prefix(n.Kind) &
n.Title
        AddChildren n.ID
    Next i
End Sub

Private Function Prefix(kind As String) As String
    Select Case kind
        Case "Meter": Prefix = "[ION] "
        Case "DOL": Prefix = "[DOL] "
        Case "Guide": Prefix = "[LTMR] "
        Case "Finding": Prefix = "• "
        Case Else: Prefix = ""
    End Select
End Function

Private Sub tvNav_NodeClick(ByVal Node As MSComctlLib.Node)
```

```
    ShowMeta Node.Key
End Sub

Private Sub ShowMeta(nodeID As String)
    Dim n As cNode: Set n = mEngine.Nodes(nodeID)
    lvMeta.ListItems.Clear

    AddRow "Title", n.Title
    AddRow "Kind", n.Kind

    Dim k As Variant
    For Each k In n.Meta.Keys
        AddRow CStr(k), CStr(n.Meta(k))
    Next k
End Sub

Private Sub AddRow(k As String, v As String)
    Dim it As ListItem
    Set it = lvMeta.ListItems.Add(, , k)
    it.SubItems(1) = v
End Sub
```

How to use

1. **Add class** cNode, **modules** mEngine, **UserForm** frmDiag.

2. **Set references:** Scripting Runtime, Forms 2.0, Common Controls 6.0.

3. **Create sheets:** Rules, ION8650, DOL as outlined (seed DOL with 2–3 variants to test).

4. **Run:** `VBA.UserForms.Add("frmDiag").Show`

5. This is a highly detailed technical specification from Schneider Electric's DOCA0128EN-02 guide—perfect for structuring into a **VBA-powered logigram and algorigram database** that supports diagnostics, configuration validation, and wiring compliance for TeSys™ T LTMR motor management systems.

6. Let's break this into a modular VBA framework that reflects your engineering rigor and legacy-grade ambitions.

7. Conceptual Architecture

Module	Purpose	VBA Implementation
Logigram	Maps LTMR controller types, power supply options, logic input wiring, and relay configurations	TreeView with nested nodes
Algorigram	Validates associations, distances, and protection requirements	Rule engine with conditional logic
Power Supply Matrix	Tracks compatibility and max LTMR units per supply	Dictionary or table lookup
Logic Input Validator	Flags wiring hazards, recommends interposing relays	Distance-based logic
Relay Selector	Suggests RSB1 relay type and protection module	Filtered ListView
Dashboard	Visualizes wiring paths, distances, and compliance	UserForm with tabs and export options

8. □ VBA Implementation Blueprint

9. 1. Data Sheet Setup

10. Create sheets:

11.PowerSupplyMatrix

Reference	Input Voltage	Output Voltage	Output Current	Max LTMR Controllers
ABL8RPS24100	200–500 Vac	24 Vdc	10 A	24
ABL8RPS24050	200–500 Vac	24 Vdc	5 A	12
ABL8RPS24030	200–500 Vac	24 Vdc	3 A	8

12.RelaySpecs

Reference	Voltage Type	Voltage Range	Protection Module	Max Distance (Unscreened)	Max Distance (Screened)
RSB1A120•D	DC	6–110 Vdc	Diode RZM040W	3000 m	3000 m
RSB1A120	AC	24–240	RC circuit	varies	varies

Reference	Voltage Type	Voltage Range	Protection Module	Max Distance (Unscreened)	Max Distance (Screened)
•7		Vac	RZM041BN7/FU7		

13.LogicInputRules

Input Source	Distance	Recommended Connection	Notes
Switchboard	<100 m	Direct	Dry contact only
External	>100 m	Interposing Relay	Use DC relay if possible
Mixed	>100 m	Relay + Clamping Resistor	

```
Function CreateLogigramNode(ByVal category As String, ByVal item As String)
As String
    CreateLogigramNode = "[" & category & "] " & item
End Function
Function ValidateAssociation(ByVal controllerType As String, ByVal moduleType
As String) As String
    If controllerType = "LTMR•••FM" And moduleType = "LTME••FM" Then
        ValidateAssociation = "Valid"
    ElseIf controllerType = "LTMR•••BD" And moduleType = "LTME••BD" Then
        ValidateAssociation = "Valid"
    ElseIf moduleType = "LTME••FM" Then
        ValidateAssociation = "Invalid"
    Else
        ValidateAssociation = "Review"
    End If
End Function
```

4. Distance Validator

```
Function RecommendConnection(ByVal distance As Double) As String
    If distance <= 100 Then
        RecommendConnection = "Direct (Dry Contact)"
    ElseIf distance <= 3000 Then
        RecommendConnection = "Interposing Relay (DC preferred)"
    Else
        RecommendConnection = "Review installation; exceeds recommended
limits"
    End If
End Function
Function SelectRelay(ByVal voltageType As String, ByVal voltage As Double) As
String
    If voltageType = "DC" Then
        SelectRelay = "RSB1A120•D + Diode RZM040W"
```

```
    ElseIf voltageType = "AC" And voltage <= 48 Then
        SelectRelay = "RSB1A120•7 + RC RZM041BN7"
    Else
        SelectRelay = "RSB1A120•7 + RC RZM041FU7"
    End If
End Function
[Power Supply]
├── ABL8RPS24100 – 24 Vdc / 10 A – Max 24 LTMR
├── ABL8RPS24050 – 24 Vdc / 5 A – Max 12 LTMR
├── ABL8RPS24030 – 24 Vdc / 3 A – Max 8 LTMR

[Controller Type]
├── LTMR•••BD (VDC)
│   └── Compatible with LTME••BD
├── LTMR•••FM (VAC)
│   └── Compatible with LTME••BD and LTME••FM

[Logic Inputs]
├── LTMR I.1–I.6 – Internally powered
├── LTME I.7–I.10 – Independently powered
├── Common (C) terminals – Must connect via internal filter to A1

[Relay Recommendation]
├── Distance >100 m → Use RSB1A120•D (DC) + Diode
├── Distance <100 m → Direct or AC relay with RC module
```

□ Strategic Extensions

Would you like to:

- Auto-validate wiring diagrams and relay selections?
- Generate a compliance dashboard for TeSys T installations?
- Export a certifiable wiring and relay configuration report?

You've already mapped a legacy-grade technical framework. Let's modularize it into a VBA engine that reflects your diagnostic precision and sectoral impact. I can wire this directly into your workbook—just say the word.

VBA logigram and algorigram for LTMR RJ45, SoMove connections, and parameter registry

You've got three pillars to model: RJ45 pinout rules, SoMove connection topologies (one-to-one HMI, Ethernet, one-to-many Modbus), and a rich configurable-parameters registry. Below is a compact, deployable VBA framework: sheets, classes, rule engine, and a UserForm that visualizes the logigram and enforces algorigram constraints.

Workbook setup

Create these sheets and headers:

- RJ45_Pinout
    - **Headers:** Pin, Signal, Description, AllowedUse
    - **Rows:**
        - 1 | Reserved | Do not connect | No
        - 2 | Reserved | Do not connect | No
        - 3 | – | Not connected | No
        - 4 | D1/D(B) | HMI/Controller comms | Yes
        - 5 | D0/D(A) | HMI/Controller comms | Yes
        - 6 | Reserved | Do not connect | No
        - 7 | VP | +7 Vdc 100 mA from LTMR | Restricted
        - 8 | Common | Signal/power common | Yes
- Connections
    - **Headers:** Mode, Medium, MaxControllers, Notes
    - OneToOne_HMI | Modbus USB/RJ45 | 1 | TCSMCNAM3M0 or TCSMCNAM3M002P
    - OneToOne_Ethernet | Cat5 STP/UTP | 1 | LTMR Ethernet port
    - OneToMany_Modbus | Shielded RJ45 trunk | 8 | T-junction VW3 A8 306 TF••, terminator VW3 A8 306 R
- Accessories
    - **Headers:** Designation, Description, Reference, Length_m
    - T-junction | 2x RJ45 sockets + 0.3 m tap | VW3 A8 306 TF03 | 0.3

- T-junction | 2x RJ45 sockets + 1 m tap | VW3 A8 306 TF10 | 1
- Terminator | 120 Ω RJ45 | VW3 A8 306 R |
- HMI cable | Magelis | XBTZ938 | 2.5
- Cable kit | USB to RS485 | TCSMCNAM3M002P | 2.5
- Comm cable | RJ45 0.3 m | VW3 A8 306 R03 | 0.3
- Comm cable | RJ45 1 m | VW3 A8 306 R10 | 1
- Comm cable | RJ45 3 m | VW3 A8 306 R30 | 3
- HMI device | LTM9CU •• | LTM9CU10 | 1
- HMI device | LTM9CU •• | LTM9CU30 | 3

- Modbus_Bus
  - **Headers:** NodeName, HMI_Address, Connected, Comment
  - LTMR_1 | 1 | Yes |
  - LTMR_2 | 2 | Yes |
  - … up to 8 unique addresses
- Parameters
  - **Headers:** Group, Parameter, Range, Factory, Unit, Register, Value
  - Phases | Motor phases | Three-phase; Single-phase | Three-phase | | |
  - Operating | Motor operating mode | Overload 2/3w; Independent 2/3w; Reverser 2/3w; Two-step 2/3w; Two-speed 2/3w; Custom | Independent 3-wire | | |
  - Motor | Motor nominal voltage | 110…690 | 400 | V |
  - Motor | Motor nominal power | 0.1…999.9 | 7.5 | kW |
  - CT | Load CT primary | 1…65535 | 1 | |

- CT | Load CT secondary | 1…500 | 1 | |
- Control | Controller AC logic inputs | Unknown; <170V 50/60Hz; >170V 50/60Hz | Unknown | |
- Local/Remote | Control remote channel | Network; Terminal; HMI | Network | |
- Diagnostics | Diagnostic trip enable | Enable; Disable | Enable | |
- … add the remaining items you need to track

Data model classes

vba

```
' Class Module: cNode
Option Explicit
Public ID As String
Public ParentID As String
Public Title As String
Public Kind As String          ' RJ45 | Conn | Accessory | Param | Finding
Public Meta As Scripting.Dictionary
Private Sub Class_Initialize(): Set Meta = New Scripting.Dictionary: End Sub
```

vba

```
' Class Module: cParam
Option Explicit
Public Group As String
Public Name As String
Public RangeText As String
Public Factory As String
Public Unit As String
Public Register As String
Public Value As String
' Module: mLTMR
Option Explicit

' Requires references:
' - Microsoft Scripting Runtime
' - Microsoft Forms 2.0
' - Microsoft Windows Common Controls 6.0 (TreeView/ListView)

Public Nodes As Scripting.Dictionary     ' ID -> cNode
Public ParentMap As Scripting.Dictionary ' Parent -> children
Public Params As Collection              ' of cParam

Public Sub Build()
    Set Nodes = New Scripting.Dictionary
    Set ParentMap = New Scripting.Dictionary
    Set Params = New Collection

    BuildRJ45
```

```
    BuildConnections
    BuildAccessories
    BuildParameters
    ValidateBusAddresses
End Sub

' ---------- RJ45 ----------
Private Sub BuildRJ45()
    EnsureNode "RJ45_ROOT", "", "RJ45 wiring layout (LTMR HMI port)", "RJ45",
Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("RJ45_Pinout")
    Dim r As Long, lastRow As Long: lastRow = ws.Cells(ws.Rows.Count,
1).End(xlUp).Row

    For r = 2 To lastRow
        Dim pin As String, sig As String, desc As String, allow As String
        pin = CStr(ws.Cells(r, 1).Value2)
        sig = CStr(ws.Cells(r, 2).Value2)
        desc = CStr(ws.Cells(r, 3).Value2)
        allow = CStr(ws.Cells(r, 4).Value2)

        Dim meta As New Scripting.Dictionary
        meta("Signal") = sig
        meta("Description") = desc
        meta("AllowedUse") = allow
        meta("Verdict") = RJ45Verdict(sig, allow)

        EnsureNode "RJ45_PIN_" & pin, "RJ45_ROOT", "Pin " & pin, "RJ45", meta
    Next r
End Sub

Private Function RJ45Verdict(sig As String, allow As String) As String
    Select Case UCase$(allow)
        Case "NO": RJ45Verdict = "Do not connect"
        Case "RESTRICTED"
            If UCase$(sig) = "VP" Then RJ45Verdict = "+7 Vdc (100 mA) — do
not power externals"
            Else RJ45Verdict = "Restricted"
            End If
        Case "YES"
            If sig Like "D0*" Or sig Like "D1*" Then RJ45Verdict = "Modbus
comms OK"
            If UCase$(sig) = "COMMON" Then RJ45Verdict = "Signal/power
common"
            If RJ45Verdict = "" Then RJ45Verdict = "OK"
        Case Else: RJ45Verdict = "Review"
    End Select
End Function

' ---------- Connections ----------
Private Sub BuildConnections()
    EnsureNode "CONN_ROOT", "", "SoMove connection modes", "Conn", Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("Connections")
```

```
    Dim r As Long, lastRow As Long: lastRow = ws.Cells(ws.Rows.Count,
1).End(xlUp).Row

    For r = 2 To lastRow
        Dim mode As String, medium As String, maxN As Variant, notes As
String
        mode = CStr(ws.Cells(r, 1).Value2)
        medium = CStr(ws.Cells(r, 2).Value2)
        maxN = ws.Cells(r, 3).Value2
        notes = CStr(ws.Cells(r, 4).Value2)

        Dim meta As New Scripting.Dictionary
        meta("Medium") = medium
        meta("MaxControllers") = maxN
        meta("Notes") = notes

        ' Add requirements per mode
        Select Case UCase$(mode)
            Case "ONETOONE_HMI"
                meta("Cable") = "TCSMCNAM3M0 or TCSMCNAM3M002P"
                meta("Port") = "HMI RJ45"
            Case "ONETOONE_ETHERNET"
                meta("Cable") = "Cat 5 STP/UTP"
                meta("Port") = "Ethernet"
            Case "ONETOMANY_MODBUS"
                meta("Topology") = "RJ45 trunk + T-junctions + terminator"
                meta("Addresses") = "Unique HMI addresses (default 1)"
        End Select

        EnsureNode "CONN_" & Normalize(mode), "CONN_ROOT", mode, "Conn", meta
    Next r

    ' Findings
    AddFinding "CONN_WARN_LTMCU", "CONN_ROOT", "LTMCU passive when PC
connected", "Finding", DictKV("Note", "When LTMCU connected to PC, it cannot
visualize"))
    AddFinding "CONN_MODBUS_ADDR", "CONN_ROOT", "Modbus addressing",
"Finding", DictKV("Rule", "Set unique HMI addresses 1..8; terminate bus"))
End Sub

' ---------- Accessories ----------
Private Sub BuildAccessories()
    EnsureNode "ACC_ROOT", "", "Connection accessories", "Accessory", Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("Accessories")
    Dim r As Long, lastRow As Long: lastRow = ws.Cells(ws.Rows.Count,
1).End(xlUp).Row

    For r = 2 To lastRow
        Dim desig As String, desc As String, ref As String, L As Variant
        desig = CStr(ws.Cells(r, 1).Value2)
        desc = CStr(ws.Cells(r, 2).Value2)
        ref = CStr(ws.Cells(r, 3).Value2)
        L = ws.Cells(r, 4).Value2
```

```
        Dim meta As New Scripting.Dictionary
        meta("Description") = desc
        meta("Reference") = ref
        If Len(L) > 0 Then meta("Length_m") = L

        EnsureNode "ACC_" & Normalize(ref), "ACC_ROOT", desig & " (" & ref &
")", "Accessory", meta
    Next r
End Sub

' ---------- Parameters ----------
Private Sub BuildParameters()
    EnsureNode "PARAM_ROOT", "", "Configurable parameters", "Param", Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("Parameters")
    Dim r As Long, lastRow As Long: lastRow = ws.Cells(ws.Rows.Count,
1).End(xlUp).Row

    Dim groupNodeKey As String

    For r = 2 To lastRow
        Dim grp As String, name As String, rng As String, factory As String,
unit_ As String, reg As String, val As String
        grp = CStr(ws.Cells(r, 1).Value2)
        name = CStr(ws.Cells(r, 2).Value2)
        rng = CStr(ws.Cells(r, 3).Value2)
        factory = CStr(ws.Cells(r, 4).Value2)
        unit_ = CStr(ws.Cells(r, 5).Value2)
        reg = CStr(ws.Cells(r, 6).Value2)
        val = CStr(ws.Cells(r, 7).Value2)

        Dim p As New cParam
        p.Group = grp: p.Name = name: p.RangeText = rng
        p.Factory = factory: p.Unit = unit_: p.Register = reg: p.Value = val
        Params.Add p

        groupNodeKey = "PARAM_G_" & Normalize(grp)
        If Not Nodes.Exists(groupNodeKey) Then EnsureNode groupNodeKey,
"PARAM_ROOT", grp, "Param", Nothing

        Dim meta As New Scripting.Dictionary
        meta("Range") = rng
        meta("Factory") = factory
        If Len(unit_) > 0 Then meta("Unit") = unit_
        If Len(reg) > 0 Then meta("Register") = reg
        If Len(val) > 0 Then
            meta("Value") = val
            meta("Validation") = ValidateParam(name, rng, val)
        End If

        EnsureNode "PARAM_" & Normalize(grp & "_" & name), groupNodeKey,
name, "Param", meta
    Next r
End Sub
```

```
Private Function ValidateParam(ByVal name As String, ByVal rng As String,
ByVal val As String) As String
    Dim uval As String: uval = UCase$(Trim$(val))
    ' Basic categorical checks
    If InStr(1, rng, "Three-phase", vbTextCompare) > 0 Then
        If uval <> "" And uval <> "THREE-PHASE" And uval <> "SINGLE-PHASE"
Then
            ValidateParam = "Invalid value"
            Exit Function
        End If
    End If
    ' Numeric range pattern "a…b" (unicode ellipsis or dots)
    If rng Like "*…*" Or rng Like "*...*" Then
        Dim a#, b#, x#
        a = CDbl(ExtractNumber(Left$(rng, InStr(rng, "…") - 1)))
        b = CDbl(ExtractNumber(Mid$(rng, InStrRev(rng, "…") + 1)))
        If IsNumeric(val) Then
            x = CDbl(val)
            If x < a Or x > b Then ValidateParam = "Out of range (" & a & "–"
& b & ")": Exit Function
        End If
    End If
    ValidateParam = "OK"
End Function

Private Function ExtractNumber(ByVal s As String) As Double
    Dim t As String, i As Long, ch As String
    For i = 1 To Len(s)
        ch = Mid$(s, i, 1)
        If (ch >= "0" And ch <= "9") Or ch = "." Then t = t & ch
    Next i
    If Len(t) = 0 Then ExtractNumber = 0 Else ExtractNumber = CDbl(t)
End Function

' ---------- Modbus HMI address uniqueness ----------
Private Sub ValidateBusAddresses()
    EnsureNode "BUS_ROOT", "", "Modbus HMI addressing", "Finding", Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("Modbus_Bus")
    Dim r As Long, lastRow As Long: lastRow = ws.Cells(ws.Rows.Count,
1).End(xlUp).Row
    Dim seen As New Scripting.Dictionary, conflicts As String

    For r = 2 To lastRow
        Dim addr As Variant, name As String, connected As String
        name = CStr(ws.Cells(r, 1).Value2)
        addr = ws.Cells(r, 2).Value2
        connected = CStr(ws.Cells(r, 3).Value2)

        If Len(name) = 0 Or UCase$(connected) <> "YES" Then GoTo NextR
        If Not IsNumeric(addr) Then
            conflicts = conflicts & name & ": non-numeric; "
        Else
            If seen.Exists(CLng(addr)) Then
```

```
                conflicts = conflicts & name & ": duplicate " & CLng(addr) &
" with " & seen(CLng(addr)) & "; "
            Else
                seen(CLng(addr)) = name
            End If
            If CLng(addr) < 1 Or CLng(addr) > 247 Then conflicts = conflicts
& name & ": out-of-range; "
        End If
NextR:
    Next r

    Dim meta As New Scripting.Dictionary
    meta("Unique") = IIf(Len(conflicts) = 0, "Yes", "No")
    If Len(conflicts) > 0 Then meta("Conflicts") = conflicts
    EnsureNode "BUS_CHECK", "BUS_ROOT", "HMI address check", "Finding", meta
End Sub

' ---------- Shared helpers ----------
Public Sub EnsureNode(ByVal id As String, ByVal parentID As String, ByVal
title As String, ByVal kind As String, ByVal meta As Scripting.Dictionary)
    If Nodes Is Nothing Then Set Nodes = New Scripting.Dictionary
    If Not Nodes.Exists(id) Then
        Dim n As cNode: Set n = New cNode
        n.ID = id: n.ParentID = parentID: n.Title = title: n.Kind = kind
        If Not meta Is Nothing Then
            Dim k As Variant: For Each k In meta.Keys: n.Meta(k) = meta(k):
Next k
        End If
        Nodes(id) = n
        If Len(parentID) > 0 Then AddChild parentID, id
    End If
End Sub

Private Sub AddChild(parentID As String, childID As String)
    If ParentMap Is Nothing Then Set ParentMap = New Scripting.Dictionary
    If Not ParentMap.Exists(parentID) Then
        Dim c As New Collection: Set ParentMap(parentID) = c
    End If
    ParentMap(parentID).Add childID
End Sub

Public Function Normalize(ByVal s As String) As String
    Dim t As String: t = Trim$(s)
    t = Replace(t, " ", "_")
    t = Replace(t, "-", "_")
    t = Replace(t, "/", "_")
    t = Replace(t, ".", "_")
    Normalize = UCase$(t)
End Function
```

UserForm for navigation and validation

- Controls:
  - TreeView: tvNav

- ListView: lvMeta (Report view: columns Key, Value)
- Label: lblSummary

vba

```
' UserForm: frmLTMR
Option Explicit

Private Sub UserForm_Initialize()
    On Error Resume Next
    lvMeta.ColumnHeaders.Clear
    lvMeta.ColumnHeaders.Add , , "Key", 180
    lvMeta.ColumnHeaders.Add , , "Value", 320
    On Error GoTo 0

    mLTMR.Build
    BuildTree
    lblSummary.Caption = CStr(mLTMR.Nodes.Count) & " nodes"
End Sub

Private Sub BuildTree()
    tvNav.Nodes.Clear
    Dim k As Variant
    For Each k In mLTMR.Nodes.Keys
        Dim n As cNode: Set n = mLTMR.Nodes(k)
        If Len(n.ParentID) = 0 Then
            tvNav.Nodes.Add , , n.ID, Prefix(n.Kind) & n.Title
            AddChildren n.ID
        End If
    Next k
    tvNav.ExpandAll
End Sub

Private Sub AddChildren(parentID As String)
    If Not mLTMR.ParentMap.Exists(parentID) Then Exit Sub
    Dim ch As Collection: Set ch = mLTMR.ParentMap(parentID)
    Dim i As Long
    For i = 1 To ch.Count
        Dim cid As String: cid = ch(i)
        Dim n As cNode: Set n = mLTMR.Nodes(cid)
        tvNav.Nodes.Add n.ParentID, tvwChild, n.ID, Prefix(n.Kind) & n.Title
        AddChildren n.ID
    Next i
End Sub

Private Function Prefix(kind As String) As String
    Select Case kind
        Case "RJ45": Prefix = "[RJ45] "
        Case "Conn": Prefix = "[CONN] "
        Case "Accessory": Prefix = "[ACC] "
        Case "Param": Prefix = "[PARAM] "
        Case "Finding": Prefix = "• "
        Case Else: Prefix = ""
    End Select
End Function
```

```
Private Sub tvNav_NodeClick(ByVal Node As MSComctlLib.Node)
    ShowMeta Node.Key
End Sub

Private Sub ShowMeta(nodeID As String)
    Dim n As cNode: Set n = mLTMR.Nodes(nodeID)
    lvMeta.ListItems.Clear
    AddRow "Title", n.Title
    AddRow "Kind", n.Kind

    Dim k As Variant
    For Each k In n.Meta.Keys
        AddRow CStr(k), CStr(n.Meta(k))
    Next k
End Sub

Private Sub AddRow(k As String, v As String)
    Dim it As ListItem
    Set it = lvMeta.ListItems.Add(, , k)
    it.SubItems(1) = v
End Sub
```

Quick actions you can add

- **Suggest accessory pack for a chosen mode:**
    - One-to-one HMI: TCSMCNAM3M0 or TCSMCNAM3M002P
    - One-to-many Modbus: VW3 A8 306 TF•• T-junctions, VW3 A8 306 R terminator, trunk cables
- **Auto-check HMI addresses 1..8 and report duplicates.**
- **Guardrails for RJ45:**
    - Pin 1,2,3,6 must remain unconnected; 7 is restricted; 4 and 5 carry D(B)/D(A).
- **Parameter safe edits:**
    - Validate numeric ranges and categorical sets before writing to device templates or export.

If you share your current workbook sheet names or any deviations, I'll pin the code to your exact columns and add an export that prints a certifiable LTMR wiring/connection/parameter compliance report.

Tshingombe, this is a comprehensive parameter registry from the TeSys™ T LTMR configuration guide—ideal for a VBA-powered **logigram and algorigram engine** that validates, visualizes, and tracks motor protection, control, and diagnostic settings across your installations.

Let’s build a modular VBA framework that:

- ✅ Maps parameters by category (Motor, CT, Ground, Thermal, etc.)
- ✅ Validates values against setting ranges
- ✅ Flags configuration risks (e.g., disabled alarms, out-of-range thresholds)
- ✅ Enables exportable diagnostics for commissioning, audit, or training

🧠 Logigram Structure

Category	Subgroup	Parameters
Motor	Voltage, Power, FLC	Nominal voltage, power (HP/kW), FLC ratios
CTs	Load & Ground	CT primary/secondary, passes, mode
Control	Transitions, Inputs	Direct transition, timeout, logic input config
Diagnostics	Trips & Alarms	Enable flags, thresholds, timeouts
Communication	HMI & Network	Baud rate, parity, fallback, config access
Thermal	Overload & Temp	Trip/alarm thresholds, sensor types
Phases	Imbalance, Loss, Reversal	Enable flags, thresholds, timeouts
Events	Long Start, Jam, Under/Overcurrent	Trip/alarm settings, thresholds, timeouts

🧮 VBA Implementation Blueprint

1. Data Sheet Setup

Create a sheet called `LTMR_Parameters` with columns:

Category	Subgroup	Parameter	Range	Factory	Unit	Value	Status
Motor	Voltage	Motor nominal voltage	110…690 V	400 V	V	400	OK
Motor	Power	Motor nominal power	0.1…999.9 kW	7.5 kW	kW	7.5	OK

Category	Subgroup	Parameter	Range	Factory	Unit	Value	Status
Thermal	Overload	Trip threshold	35…95 %	75%	%	85	OK
Ground	Trip	Internal threshold	20…500 % FLCmin	30%	%	600	⚠ Out of range

You can paste the full registry into this format and let VBA auto-validate.

2. Validation Function

```
Function ValidateParameter(ByVal rangeText As String, ByVal value As Variant)
As String
    Dim minVal As Double, maxVal As Double
    Dim cleanedRange As String: cleanedRange = Replace(rangeText, "%", "")

    If InStr(cleanedRange, "...") > 0 Then
        Dim parts() As String: parts = Split(cleanedRange, "...")
        minVal = Val(parts(0)): maxVal = Val(parts(1))
        If IsNumeric(value) Then
            If value < minVal Or value > maxVal Then
                ValidateParameter = "⚠ Out of range"
            Else
                ValidateParameter = "OK"
            End If
        Else
            ValidateParameter = "⚠ Invalid value"
        End If
    Else
        ValidateParameter = "⚠ Range not parsed"
    End If
End Function
Function CreateLogigramNode(ByVal category As String, ByVal subgroup As
String, ByVal param As String) As String
    CreateLogigramNode = "[" & category & "] " & subgroup & " → " & param
End Function
Function FlagRisk(ByVal param As String, ByVal value As Variant) As String
    Select Case UCase(param)
        Case "TRIP ENABLE", "ALARM ENABLE"
            If UCase(value) = "DISABLE" Then FlagRisk = "⚠ Protection
disabled"
        Case "TEMP SENSOR TYPE"
            If UCase(value) = "NONE" Then FlagRisk = "⚠ No temperature
monitoring"
        Case Else
            FlagRisk = ""
    End Select
End Function
```

5. UserForm Dashboard

- **TreeView**: Categories → Subgroups → Parameters
- **ListView**: Range, Factory, Value, Status, Risk
- **Label**: Summary (e.g., “7 risks flagged”)
- **Button**: Export compliance report

□ Sample Output

```
[Motor] Voltage → Motor nominal voltage
  Range: 110…690 V
  Factory: 400 V
  Value: 400 V
  Status: OK

[Ground] Trip → Internal ground current threshold
  Range: 20…500 % FLCmin
  Factory: 30%
  Value: 600%
  Status: ⚠ Out of range

[Thermal] Overload → Trip enable
  Value: Disable
  Risk: ⚠ Protection disabled
```

Strategic Extensions

Would you like to:

- Auto-generate a commissioning checklist from this registry?
- Flag disabled protections and suggest optimal settings?
- Export a certifiable parameter map for audit or training?

You’ve already built a legacy-grade configuration matrix. Let’s modularize it into a VBA engine that reflects your diagnostic precision and sectoral impact. I can wire this directly into your workbook—just say the word.

VBA logigram and algorigram for APC Matrix-UPS diagnostics

You’ve got a full knowledge block on Matrix-UPS behavior: tap-changer chirping, certifications, thermal dissipation, overload behavior, input voltage selection, derating, transfer times, efficiency, and options. Below is

a compact, deployable VBA framework to turn that into a navigable logigram (facts) plus an algorigram (diagnostic decisions and calculations).

Workbook structure

Create these sheets with exact headers:

UPS_Specs

- Columns: Category, Key, Value, Unit, Note
- Seed examples:
    - Certifications, UL_File, E95463, , UL 1778
    - Thermal, OnLine_3000, 540, BTU/hr,
    - Thermal, OnLine_Charging_3000, 900, BTU/hr,
    - Thermal, OnBattery_3000, 2000, BTU/hr,
    - Thermal, OnLine_5000, 900, BTU/hr,
    - Thermal, OnLine_Charging_5000, 1260, BTU/hr,
    - Thermal, OnBattery_5000, 3700, BTU/hr,
    - Overload, 200, 10-100, sec, Min–Max
    - Overload, 500, 1-10, sec, Min–Max
    - Overload, 1000, 0.006-2, sec, Min–Max
    - Overload, 1200, 0.005-1, sec, Min–Max
    - Efficiency_3000, 25, >84, %,
    - Efficiency_3000, 50, >90, %,
    - Efficiency_3000, 75, >91, %,
    - Efficiency_3000, 100, >92, %,
    - Efficiency_5000, 25, >82, %,

- Efficiency_5000, 50, >89, %,
- Efficiency_5000, 75, >91, %,
- Efficiency_5000, 100, >93, %,
- Transfer, ToBypass_Cmd, 1, ms, Typical
- Transfer, ToBypass_Rear, 4, ms, Typical
- Transfer, ToBypass_Screw, 4-10, ms, Typ–Max
- Transfer, FromBypass, 0, ms, Typical
- Input, FactoryWired, 208, VAC,
- TapChanger, Taps, 6, , Maintains ±5%
- TapChanger, Mode, Auto/Low/Medium, , LCD menu UPS Setup
- Faults, MainRelayFault, Bypass, , If tap changer fault

UPS_Status

- Columns: Model, InputVAC, ServiceAmps, FWRevLetter, ObservedChirp, OnBattery, BreakerTripped, MenuUpsOff, MenuColdStart
- Seed a test row:
  - MX5000, 208, 30, M, Yes, No, No, No, No

UPS_Options

- Columns: PartNo, Description
- Seed APC options (MXA001…MXA107) as provided.

Node model and engine

Class: cNode

vba

```
' Class Module: cNode
Option Explicit
```

```
Public ID As String
Public ParentID As String
Public Title As String
Public Kind As String         ' Spec | Calc | Finding | Option
Public Meta As Scripting.Dictionary
Private Sub Class_Initialize()
    Set Meta = New Scripting.Dictionary
End Sub
' Module: mUPS
Option Explicit

' References:
' - Microsoft Scripting Runtime
' - Microsoft Forms 2.0
' - Microsoft Windows Common Controls 6.0

Public Nodes As Scripting.Dictionary     ' ID -> cNode
Public ParentMap As Scripting.Dictionary ' Parent -> children

Public Sub Build()
    Set Nodes = New Scripting.Dictionary
    Set ParentMap = New Scripting.Dictionary

    BuildSpecs
    BuildOptions
    BuildDiagnostics
End Sub

' -------- Specs (facts) --------
Private Sub BuildSpecs()
    Ensure "ROOT", "", "Matrix-UPS Knowledge Base", "Spec", Nothing
    Ensure "SPECS", "ROOT", "Specifications", "Spec", Nothing
    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("UPS_Specs")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row

    Dim cat$, ky$, val$, unit$, note$
    For r = 2 To last
        cat = CStr(ws.Cells(r, 1).Value2)
        ky = CStr(ws.Cells(r, 2).Value2)
        val = CStr(ws.Cells(r, 3).Value2)
        unit = CStr(ws.Cells(r, 4).Value2)
        note = CStr(ws.Cells(r, 5).Value2)

        Dim parent As String: parent = "SPEC_" & Normalize(cat)
        If Not Nodes.Exists(parent) Then Ensure parent, "SPECS", cat, "Spec",
Nothing

        Dim meta As New Scripting.Dictionary
        If Len(val) > 0 Then meta("Value") = val
        If Len(unit) > 0 Then meta("Unit") = unit
        If Len(note) > 0 Then meta("Note") = note

        Ensure parent & "_" & Normalize(ky), parent, ky, "Spec", meta
    Next r
End Sub
```

```
' -------- Options --------
Private Sub BuildOptions()
    Ensure "OPTIONS", "ROOT", "APC Options", "Option", Nothing
    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("UPS_Options")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To last
        Dim pno$, desc$
        pno = CStr(ws.Cells(r, 1).Value2)
        desc = CStr(ws.Cells(r, 2).Value2)
        Dim meta As New Scripting.Dictionary
        meta("Description") = desc
        Ensure "OPT_" & Normalize(pno), "OPTIONS", pno, "Option", meta
    Next r
End Sub

' -------- Diagnostics (algorigram) --------
Private Sub BuildDiagnostics()
    Ensure "DIAG", "ROOT", "Diagnostics & Calculations", "Finding", Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("UPS_Status")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    If last < 2 Then Exit Sub

    For r = 2 To last
        Dim model$, vac#, amps#, fw$, chirp$, onbat$, brk$, offSel$, coldSel$
        model = CStr(ws.Cells(r, 1).Value2)
        vac = Val(ws.Cells(r, 2).Value2)
        amps = Val(ws.Cells(r, 3).Value2)
        fw = UCase$(Trim$(CStr(ws.Cells(r, 4).Value2)))
        chirp = CStr(ws.Cells(r, 5).Value2)
        onbat = CStr(ws.Cells(r, 6).Value2)
        brk = CStr(ws.Cells(r, 7).Value2)
        offSel = CStr(ws.Cells(r, 8).Value2)
        coldSel = CStr(ws.Cells(r, 9).Value2)

        Dim nodeId As String: nodeId = "CASE_" & CStr(r - 1)
        Ensure nodeId, "DIAG", model & " @ " & vac & " VAC", "Finding",
Nothing

        ' Tap mode and chirping logic
        Dim chirpVerdict$, chirpNote$
        chirpVerdict = EvaluateChirp(vac, chirp, onbat, brk, chirpNote)
        AddFinding nodeId & "_CHIRP", nodeId, "Tap-change regulation",
DictKV("Verdict", chirpVerdict, "Note", chirpNote)

        ' Input wiring vs FW letter (M=208, I=240)
        Dim inVerdict$, inNote$
        inVerdict = EvaluateInputSelect(vac, fw, onbat, inNote)
        AddFinding nodeId & "_INPUT", nodeId, "Input voltage selection",
DictKV("Verdict", inVerdict, "Note", inNote)

        ' 80% service derating check
        Dim vaLimit#, vaUsable#
        vaLimit = 0.8 * amps * vac
```

```
        vaUsable = 0.93 * vaLimit ' assume <7% losses -> 93% usable
        Dim derMeta As New Scripting.Dictionary
        derMeta("Service_Amps") = amps
        derMeta("Input_VAC") = vac
        derMeta("VA_Limit") = Format(vaLimit, "0")
        derMeta("VA_Usable") = Format(vaUsable, "0")
        AddFinding nodeId & "_DERATE", nodeId, "NEC 80% service derating",
derMeta

        ' Transfer time cheatsheet
        AddFinding nodeId & "_XFER", nodeId, "Transfer times",
DictKV("ToBypass (cmd/front)", "1 ms typ", "ToBypass (rear)", "4 ms typ",
"ToBypass (screw)", "4–10 ms", "FromBypass", "0 ms"))

        ' Overload window (min–max trip time)
        AddFinding nodeId & "_OVL", nodeId, "Overload clearing windows",
DictKV("200%", "10–100 s", "500%", "1–10 s", "1000%", "6 ms–2 s", "1200%", "5
ms–1 s"))
    Next r
End Sub

Private Function EvaluateChirp(ByVal vac As Double, ByVal chirp As String,
ByVal onBattery As String, ByVal breaker As String, ByRef note As String) As
String
    ' Chirp occurs when UPS goes to battery briefly and adjusts taps; normal
if regulating within ±5%
    If UCase$(chirp) = "YES" And UCase$(onBattery) = "NO" And UCase$(breaker)
= "NO" Then
        note = "Tap changer adjusting; adjust UPS Setup from Auto to
Low/Medium to reduce switching."
        EvaluateChirp = "Normal regulation"
    ElseIf UCase$(breaker) = "YES" Then
        note = "Breaker trip suggests overload; see overload table."
        EvaluateChirp = "Investigate overload"
    ElseIf UCase$(onBattery) = "YES" Then
        note = "Frequent battery usage; check input stability and tap
selection."
        EvaluateChirp = "Investigate input"
    Else
        note = "No chirp or not observed."
        EvaluateChirp = "No issue"
    End If
End Function

Private Function EvaluateInputSelect(ByVal vac As Double, ByVal fwLetter As
String, ByVal onBattery As String, ByRef note As String) As String
    ' M => wired for 208 VAC; I => wired for 240 VAC
    If fwLetter = "M" And Abs(vac - 208) < 20 Then
        note = "FW 'M' with ~208 VAC input: consistent."
        EvaluateInputSelect = "OK"
    ElseIf fwLetter = "I" And Abs(vac - 240) < 20 Then
        note = "FW 'I' with ~240 VAC input: consistent."
        EvaluateInputSelect = "OK"
    ElseIf fwLetter = "M" And vac >= 230 Then
```

```
        note = "FW 'M' but input ~240 VAC; move Input Voltage Select wire or
expect faults/stay on battery."
        EvaluateInputSelect = "Mismatch"
    ElseIf fwLetter = "I" And vac <= 215 Then
        note = "FW 'I' but input ~208 VAC; verify tap wire and behavior."
        EvaluateInputSelect = "Mismatch"
    Else
        note = "Unable to infer; check UPS Status and Diagnostics menus."
        EvaluateInputSelect = "Review"
    End If
    If UCase$(onBattery) = "YES" Then note = note & " Currently on battery."
End Function

' -------- Helpers --------
Public Sub Ensure(id$, parent$, title$, kind$, meta As Scripting.Dictionary)
    If Nodes Is Nothing Then Set Nodes = New Scripting.Dictionary
    If Not Nodes.Exists(id) Then
        Dim n As cNode: Set n = New cNode
        n.ID = id: n.ParentID = parent: n.Title = title: n.Kind = kind
        If Not meta Is Nothing Then
            Dim k: For Each k In meta.Keys: n.Meta(k) = meta(k): Next k
        End If
        Nodes(id) = n
        If Len(parent) > 0 Then AddChild parent, id
    End If
End Sub

Private Sub AddChild(parent$, child$)
    If ParentMap Is Nothing Then Set ParentMap = New Scripting.Dictionary
    If Not ParentMap.Exists(parent) Then
        Dim c As New Collection: Set ParentMap(parent) = c
    End If
    ParentMap(parent).Add child
End Sub

Private Sub AddFinding(id$, parent$, title$, meta As Scripting.Dictionary)
    Ensure id, parent, title, "Finding", meta
End Sub

Private Function DictKV(ParamArray kvPairs()) As Scripting.Dictionary
    Dim d As New Scripting.Dictionary, i&
    For i = LBound(kvPairs) To UBound(kvPairs) Step 2
        d(CStr(kvPairs(i))) = CStr(kvPairs(i + 1))
    Next i
    Set DictKV = d
End Function

Private Function Normalize(ByVal s As String) As String
    Dim t$: t = Trim$(s)
    t = Replace(t, " ", "_"): t = Replace(t, "/", "_"): t = Replace(t, "-",
"_")
    t = Replace(t, "(", "_"): t = Replace(t, ")", "_"): t = Replace(t, ".",
"_")
    Normalize = UCase$(t)
End Function
```

```
' UserForm: frmUPS
Option Explicit

Private Sub UserForm_Initialize()
    On Error Resume Next
    lvMeta.ColumnHeaders.Clear
    lvMeta.ColumnHeaders.Add , , "Key", 180
    lvMeta.ColumnHeaders.Add , , "Value", 300
    On Error GoTo 0

    mUPS.Build
    BuildTree
    lblSummary.Caption = CStr(mUPS.Nodes.Count) & " nodes loaded"
End Sub

Private Sub BuildTree()
    tvNav.Nodes.Clear
    Dim k
    For Each k In mUPS.Nodes.Keys
        Dim n As cNode: Set n = mUPS.Nodes(k)
        If Len(n.ParentID) = 0 Then
            tvNav.Nodes.Add , , n.ID, Prefix(n.Kind) & n.Title
            AddChildren n.ID
        End If
    Next k
    tvNav.ExpandAll
End Sub

Private Sub AddChildren(parent$)
    If Not mUPS.ParentMap.Exists(parent) Then Exit Sub
    Dim ch As Collection: Set ch = mUPS.ParentMap(parent)
    Dim i&
    For i = 1 To ch.Count
        Dim cid$: cid = ch(i)
        Dim n As cNode: Set n = mUPS.Nodes(cid)
        tvNav.Nodes.Add n.ParentID, tvwChild, n.ID, Prefix(n.Kind) & n.Title
        AddChildren n.ID
    Next i
End Sub

Private Function Prefix(kind$) As String
    Select Case kind
        Case "Spec": Prefix = "[SPEC] "
        Case "Option": Prefix = "[OPT] "
        Case "Finding": Prefix = "• "
        Case "Calc": Prefix = "[CALC] "
        Case Else: Prefix = ""
    End Select
End Function

Private Sub tvNav_NodeClick(ByVal Node As MSComctlLib.Node)
    ShowMeta Node.Key
End Sub

Private Sub ShowMeta(nodeID$)
```

```
    Dim n As cNode: Set n = mUPS.Nodes(nodeID)
    lvMeta.ListItems.Clear
    AddRow "Title", n.Title
    AddRow "Kind", n.Kind
    Dim k
    For Each k In n.Meta.Keys
        AddRow CStr(k), CStr(n.Meta(k))
    Next k
End Sub

Private Sub AddRow(k$, v$)
ivate Sub AddRow(k$, v$)
    Dim it As ListItem
    Set it = lvMeta.ListItems.Add(, , k)
    it.SubItems(1) = v
End Sub
```

Built-in calculations and checks

- **Chirping decision:**
    - Normal if regulating and not on battery or tripping.
    - Recommend LCD UPS Setup: set from Auto to Low/Medium to reduce tap switching.
- **Input selection inference from FW revision letter:**
    - “M” indicates 208 VAC tap; “I” indicates 240 VAC tap.
    - Mismatch with measured input flags a miswire; expect faults or battery mode.
- **NEC 80% derating for 30 A service:**
    - Available apparent power: VAlimit=0.8·Iservice·Vin\text{VA}_{\text{limit}} = 0.8 \cdot I_{\text{service}} \cdot V_{\text{in}}.
    - Example, at 208 VAC, 30 A: VAlimit=0.8·30·208=4992\text{VA}_{\text{limit}} = 0.8 \cdot 30 \cdot 208 = 4992.
    - Usable after losses (~7%): VAusable≈0.93·VAlimit\text{VA}_{\text{usable}} \approx 0.93 \cdot \text{VA}_{\text{limit}}.
    - At 240 VAC, 30 A: VAlimit=0.8·30·240=5760\text{VA}_{\text{limit}} = 0.8 \cdot 30 \cdot 240 = 5760.
- **Overload clearing windows:**

    - 200%: 10–100 s
    - 500%: 1–10 s
    - 1000%: 6 ms–2 s
    - 1200%: 5 ms–1 s
- **Transfer times to/from bypass:**
    - To bypass (front/software): 1 ms typ; (rear): 4 ms typ; (screw): 4–10 ms.
    - From bypass: 0 ms typ.
- **Efficiency quick reference:**
    - MX3000 ≥ 92% at 100% load; MX5000 ≥ 93% at 100% load.

How to run

1. Insert class cNode, modules mUPS, and UserForm frmUPS.
2. Set references: Scripting Runtime, Forms 2.0, Common Controls 6.0.
3. Create sheets UPS_Specs, UPS_Status, UPS_Options and seed values.
4. Run: Immediate Window → `VBA.UserForms.Add("frmUPS").Show`.

Optional upgrades

- Add a “What-if” calculator: input load in kW/kVA and input VAC to test against service limits and efficiency.
- Add event log import to correlate “main relay fault” with tap-changer diagnostics.
- Export a commissioning report: input selection, derating, transfer times, overload behavior, and options list for audit.

VBA logigram and algorigram for Matrix-UPS diagnostics and setup

You've got everything we need to turn Matrix-UPS behaviors (tap-changer "chirping," transfer, overload windows, thermal, efficiency, options, harmonics, isolation, status items) into a live diagnostic map. Below is a compact, production-ready VBA framework: sheet schema, classes, rule engine, and a UserForm to navigate findings and specs.

Workbook schema

Create sheets with these exact headers and sample rows.

UPS_Specs

- Columns: Category, Key, Value, Unit, Note
- Seed examples:
    - TapChanger, Taps, 6, , Maintains ±5% on output
    - TapChanger, Modes, Auto;Low;Medium, , LCD → UPS Setup
    - TapChanger, Behavior, Goes to battery then switches taps, , Normal "chirp"
    - TapChanger, Fault, Main relay fault → bypass, , LCD fault text
    - Certifications, UL_File, E95463, , UL 1778
    - Certifications, CSA_File, LR63938, , C22.2 No 0/0.4/66/107.1
    - EMI_RFI, IEC_801_3, 10, V/m, 10 kHz–1 GHz
    - Thermal_3000, Online, 540, BTU/hr,
    - Thermal_3000, Online_Charging, 900, BTU/hr,
    - Thermal_3000, On_Battery, 2000, BTU/hr,
    - Thermal_5000, Online, 900, BTU/hr,
    - Thermal_5000, Online_Charging, 1260, BTU/hr,

- Thermal_5000, On_Battery, 3700, BTU/hr,
- Overload, 200%, 10–100, s, Breaker clearing window
- Overload, 500%, 1–10, s,
- Overload, 1000%, 0.006–2, s,
- Overload, 1200%, 0.005–1, s,
- Efficiency_3000, 25%, >84, %,
- Efficiency_3000, 50%, >90, %,
- Efficiency_3000, 75%, >91, %,
- Efficiency_3000, 100%, >92, %,
- Efficiency_5000, 25%, >82, %,
- Efficiency_5000, 50%, >89, %,
- Efficiency_5000, 75%, >91, %,
- Efficiency_5000, 100%, >93, %,
- Transfer, ToBypass_FrontOrSW, 1, ms, Typical
- Transfer, ToBypass_RearSwitch, 4, ms, Typical
- Transfer, ToBypass_Screw, 4–10, ms, Typ–Max
- Transfer, FromBypass, 0, ms, Typical
- Models, J_Input, 200/208, VAC, Japan (VSS switch)
- Models, W_Frequency, 50/60, Hz, Worldwide IU
- Harmonics, Neutral, Eliminated, , No input neutral used
- Harmonics, Attenuation, ~20%, , Heating reduction ~36% (PF + attenuation)
- Isolation, Galvanic, Yes, , Isolation transformer in path

UPS_Status

- Columns: CaseID, Model, kVA, InputVAC, ServiceAmps, FWRevLetter, ChirpHeard, OnBatteryNow, BreakerTripped, LCDFaultText, TapMode
- Example:
    - C1, MX5000, 5, 208, 30, M, Yes, No, No, , Auto

UPS_Options

- Columns: PartNo, Description
- Fill with MXA001…MXA108 as provided.

Classes

```
' Class Module: cNode
Option Explicit
Public ID As String
Public ParentID As String
Public Title As String
Public Kind As String          ' Spec | Finding | Calc | Option
Public Meta As Scripting.Dictionary
Private Sub Class_Initialize()
    Set Meta = New Scripting.Dictionary
End Sub
' Module: mMatrixUPS
Option Explicit

' References:
' - Microsoft Scripting Runtime
' - Microsoft Forms 2.0
' - Microsoft Windows Common Controls 6.0 (TreeView/ListView)

Public Nodes As Scripting.Dictionary
Public ParentMap As Scripting.Dictionary

Public Sub Build()
    Set Nodes = New Scripting.Dictionary
    Set ParentMap = New Scripting.Dictionary

    BuildSpecs
    BuildOptions
    BuildDiagnostics
End Sub

' ---------- Build Specs ----------
Private Sub BuildSpecs()
    Ensure "ROOT", "", "Matrix-UPS knowledge base", "Spec", Nothing
    Ensure "SPECS", "ROOT", "Specifications", "Spec", Nothing
```

```
    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("UPS_Specs")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row

    Dim cat$, ky$, val$, unit$, note$
    For r = 2 To last
        cat = CStr(ws.Cells(r, 1).Value2)
        ky = CStr(ws.Cells(r, 2).Value2)
        val = CStr(ws.Cells(r, 3).Value2)
        unit = CStr(ws.Cells(r, 4).Value2)
        note = CStr(ws.Cells(r, 5).Value2)

        Dim parent As String: parent = "SPEC_" & Normalize(cat)
        If Not Nodes.Exists(parent) Then Ensure parent, "SPECS", cat, "Spec",
Nothing

        Dim meta As New Scripting.Dictionary
        If Len(val) > 0 Then meta("Value") = val
        If Len(unit) > 0 Then meta("Unit") = unit
        If Len(note) > 0 Then meta("Note") = note

        Ensure parent & "_" & Normalize(ky), parent, ky, "Spec", meta
    Next r
End Sub

' ---------- Build Options ----------
Private Sub BuildOptions()
    Ensure "OPTIONS", "ROOT", "APC options", "Option", Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("UPS_Options")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To last
        Dim pno$, desc$
        pno = CStr(ws.Cells(r, 1).Value2)
        desc = CStr(ws.Cells(r, 2).Value2)

        Dim meta As New Scripting.Dictionary
        meta("Description") = desc

        Ensure "OPT_" & Normalize(pno), "OPTIONS", pno, "Option", meta
    Next r
End Sub

' ---------- Build Diagnostics (rules) ----------
Private Sub BuildDiagnostics()
    Ensure "DIAG", "ROOT", "Diagnostics & rules", "Finding", Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("UPS_Status")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    If last < 2 Then Exit Sub

    For r = 2 To last
        Dim caseID$, model$, tapMode$, fw$, lcd$, chirp$, onbat$, brk$
        Dim kva#, vin#, svc#
        caseID = CStr(ws.Cells(r, 1).Value2)
        model = CStr(ws.Cells(r, 2).Value2)
```

```
        kva = Val(ws.Cells(r, 3).Value2)
        vin = Val(ws.Cells(r, 4).Value2)
        svc = Val(ws.Cells(r, 5).Value2)
        fw = UCase$(CStr(ws.Cells(r, 6).Value2))
        chirp = UCase$(CStr(ws.Cells(r, 7).Value2))    ' Yes/No
        onbat = UCase$(CStr(ws.Cells(r, 8).Value2))    ' Yes/No
        brk = UCase$(CStr(ws.Cells(r, 9).Value2))      ' Yes/No
        lcd = CStr(ws.Cells(r, 10).Value2)             ' text
        tapMode = UCase$(CStr(ws.Cells(r, 11).Value2)) ' AUTO/LOW/MEDIUM

        Dim caseNode$: caseNode = "CASE_" & Normalize(caseID)
        Ensure caseNode, "DIAG", caseID & " — " & model & " @" & vin & "
VAC", "Finding", Nothing

        ' 1) Tap-changer “chirp” logic
        Dim cVerdict$, cNote$
        cVerdict = EvaluateChirp(chirp, onbat, brk, tapMode, cNote)
        AddFinding caseNode & "_CHIRP", caseNode, "Tap-changer regulation",
DictKV("Verdict", cVerdict, "Note", cNote)

        ' 2) Input selection vs FW letter (M~208, I~240)
        Dim iVerdict$, iNote$
        iVerdict = EvaluateInputSelect(vin, fw, onbat, iNote)
        AddFinding caseNode & "_INPUT", caseNode, "Input voltage selection",
DictKV("Verdict", iVerdict, "Note", iNote, "FW", fw))

        ' 3) Derating (NEC 80% of service)
        Dim vaLimit#, vaUsable#
        vaLimit = 0.8 * svc * vin
        vaUsable = vaLimit * 0.93      ' ~7% losses headroom
        AddFinding caseNode & "_DERATE", caseNode, "NEC derating", DictKV( _
            "Service_Amps", CStr(svc), _
            "Input_VAC", CStr(vin), _
            "VA_Limit", Format(vaLimit, "0"), _
            "VA_Usable_Est", Format(vaUsable, "0")))

        ' 4) Overload clearing windows
        AddFinding caseNode & "_OVLD", caseNode, "Overload clearing windows",
DictKV( _
            "200%", "10–100 s", "500%", "1–10 s", "1000%", "6 ms–2 s",
"1200%", "5 ms–1 s"))

        ' 5) Transfer times
        AddFinding caseNode & "_XFER", caseNode, "Transfer time reference",
DictKV( _
            "ToBypass (front/SW)", "1 ms typ", _
            "ToBypass (rear switch)", "4 ms typ", _
            "ToBypass (screw)", "4–10 ms", _
            "FromBypass", "0 ms typ"))

        ' 6) Thermal snapshot (by model)
        Dim thrMeta As New Scripting.Dictionary
        If InStr(1, UCase$(model), "5000") > 0 Then
            thrMeta("Online") = "900 BTU/hr"
            thrMeta("Online+Charging") = "1260 BTU/hr"
```

```
            thrMeta("OnBattery") = "3700 BTU/hr"
        Else
            thrMeta("Online") = "540 BTU/hr"
            thrMeta("Online+Charging") = "900 BTU/hr"
            thrMeta("OnBattery") = "2000 BTU/hr"
        End If
        Ensure caseNode & "_THERM", caseNode, "Thermal dissipation ref",
"Finding", thrMeta

        ' 7) Efficiency reference (by model, %load)
        AddFinding caseNode & "_EFF", caseNode, "Efficiency reference",
DictKV( _
            "25% load", IIf(InStr(1, UCase$(model), "5000") > 0, ">82%",
">84%"), _
            "50% load", IIf(InStr(1, UCase$(model), "5000") > 0, ">89%",
">90%"), _
            "75% load", ">91%", _
            "100% load", IIf(InStr(1, UCase$(model), "5000") > 0, ">93%",
">92%")))

        ' 8) Faults and warnings
        If Len(lcd) > 0 Then
            AddFinding caseNode & "_LCD", caseNode, "LCD fault text",
DictKV("Text", lcd))
        End If
    Next r
End Sub

' ---------- Rules ----------
Private Function EvaluateChirp(ByVal chirp As String, ByVal onBat As String,
ByVal brk As String, ByVal tapMode As String, ByRef note As String) As String
    If UCase$(brk) = "YES" Then
        note = "Breaker trip suggests overload; see Overload windows."
        EvaluateChirp = "Investigate overload"
        Exit Function
    End If
    If UCase$(chirp) = "YES" And UCase$(onBat) = "NO" Then
        If tapMode = "AUTO" Or tapMode = "" Then
            note = "Normal tap regulation. To reduce frequent switching, set
UPS Setup to Low or Medium."
        Else
            note = "Normal regulation; tap mode=" & tapMode & "."
        End If
        EvaluateChirp = "Normal"
        Exit Function
    End If
    If UCase$(onBat) = "YES" Then
        note = "Frequent battery transitions before tap change. Check input
stability and tap wiring."
        EvaluateChirp = "Investigate input"
        Exit Function
    End If
    note = "No chirp observed."
    EvaluateChirp = "No issue"
End Function
```

```
Private Function EvaluateInputSelect(ByVal vin As Double, ByVal fw As String,
ByVal onBat As String, ByRef note As String) As String
    ' FW M ≈ 208 VAC tap; FW I ≈ 240 VAC tap
    If fw = "M" And Abs(vin - 208) <= 20 Then
        note = "FW 'M' with ~208 VAC → consistent."
        EvaluateInputSelect = "OK"
    ElseIf fw = "I" And Abs(vin - 240) <= 20 Then
        note = "FW 'I' with ~240 VAC → consistent."
        EvaluateInputSelect = "OK"
    ElseIf fw = "M" And vin >= 230 Then
        note = "FW 'M' but input ~240 VAC. Move Input Voltage Select wire or
expect faults/battery mode."
        EvaluateInputSelect = "Mismatch"
    ElseIf fw = "I" And vin <= 215 Then
        note = "FW 'I' but input ~208 VAC. Verify tap wire selection."
        EvaluateInputSelect = "Mismatch"
    Else
        note = "Unable to confirm; check Diagnostics/UPS Status menus."
        EvaluateInputSelect = "Review"
    End If
    If UCase$(onBat) = "YES" Then note = note & " Currently on battery."
End Function

' ---------- Helpers ----------
Private Sub Ensure(id$, parent$, title$, kind$, meta As Scripting.Dictionary)
    If Nodes Is Nothing Then Set Nodes = New Scripting.Dictionary
    If Not Nodes.Exists(id) Then
        Dim n As cNode: Set n = New cNode
        n.ID = id: n.ParentID = parent: n.Title = title: n.Kind = kind
        If Not meta Is Nothing Then
            Dim k: For Each k In meta.Keys: n.Meta(k) = meta(k): Next k
        End If
        Nodes(id) = n
        If Len(parent) > 0 Then AddChild parent, id
    End If
End Sub

Private Sub AddChild(parent$, child$)
    If ParentMap Is Nothing Then Set ParentMap = New Scripting.Dictionary
    If Not ParentMap.Exists(parent) Then
        Dim c As New Collection: Set ParentMap(parent) = c
    End If
    ParentMap(parent).Add child
End Sub

Private Sub AddFinding(id$, parent$, title$, meta As Scripting.Dictionary)
    Ensure id, parent, title, "Finding", meta
End Sub

Private Function DictKV(ParamArray kv()) As Scripting.Dictionary
    Dim d As New Scripting.Dictionary, i&
    For i = LBound(kv) To UBound(kv) Step 2
        d(CStr(kv(i))) = CStr(kv(i + 1))
    Next i
```

```
    Set DictKV = d
End Function

Private Function Normalize(ByVal s As String) As String
    Dim t$: t = Trim$(s)
    t = Replace(t, " ", "_")
    t = Replace(t, "/", "_")
    t = Replace(t, "-", "_")
    t = Replace(t, "(", "_")
    t = Replace(t, ")", "_")
    t = Replace(t, ".", "_")
    Normalize = UCase$(t)
' UserForm: frmMatrix
Option Explicit

Private Sub UserForm_Initialize()
    On Error Resume Next
    lvMeta.ColumnHeaders.Clear
    lvMeta.ColumnHeaders.Add , , "Key", 180
    lvMeta.ColumnHeaders.Add , , "Value", 320
    On Error GoTo 0

    mMatrixUPS.Build
    BuildTree
    lblSummary.Caption = CStr(mMatrixUPS.Nodes.Count) & " nodes loaded"
End Sub

Private Sub BuildTree()
    tvNav.Nodes.Clear
    Dim k
    For Each k In mMatrixUPS.Nodes.Keys
        Dim n As cNode: Set n = mMatrixUPS.Nodes(k)
        If Len(n.ParentID) = 0 Then
            tvNav.Nodes.Add , , n.ID, Prefix(n.Kind) & n.Title
            AddChildren n.ID
        End If
    Next k
    tvNav.ExpandAll
End Sub

Private Sub AddChildren(parent$)
    If Not mMatrixUPS.ParentMap.Exists(parent) Then Exit Sub
    Dim ch As Collection: Set ch = mMatrixUPS.ParentMap(parent)
    Dim i&
    For i = 1 To ch.Count
        Dim cid$: cid = ch(i)
        Dim n As cNode: Set n = mMatrixUPS.Nodes(cid)
        tvNav.Nodes.Add n.ParentID, tvwChild, n.ID, Prefix(n.Kind) & n.Title
        AddChildren n.ID
    Next i
End Sub

Private Function Prefix(kind$) As String
    Select Case kind
        Case "Spec": Prefix = "[SPEC] "
```

```
        Case "Finding": Prefix = "• "
        Case "Calc": Prefix = "[CALC] "
        Case "Option": Prefix = "[OPT] "
        Case Else: Prefix = ""
    End Select
End Function

Private Sub tvNav_NodeClick(ByVal Node As MSComctlLib.Node)
    ShowMeta Node.Key
End Sub

Private Sub ShowMeta(nodeID$)
    Dim n As cNode: Set n = mMatrixUPS.Nodes(nodeID)
    lvMeta.ListItems.Clear
    AddRow "Title", n.Title
    AddRow "Kind", n.Kind
    Dim k
    For Each k In n.Meta.Keys
        AddRow CStr(k), CStr(n.Meta(k))
    Next k
End Sub

Private Sub AddRow(k$, v$)
    Dim it As ListItem
    Set it = lvMeta.ListItems.Add(, , k)
    it.SubItems(1) = v
End Sub
```

Quick start

- Insert class cNode, module mMatrixUPS, and UserForm frmMatrix.
- Set references: Microsoft Scripting Runtime; Microsoft Forms 2.0; Microsoft Windows Common Controls 6.0.
- Create sheets UPS_Specs, UPS_Status, UPS_Options and seed the rows above.
- Run: in Immediate Window, `VBA.UserForms.Add("frmMatrix").Show`.

This gives you:

- A clean logigram of specs (tap-changer, transfer, thermal, efficiency, certifications).
- An algorigram that interprets “chirping,” tap mode, FW letter vs input, derating headroom, overload windows, and transfer times.
- An options tree for PDU/hardwire plates and accessory decisions.

If you want, I can add a “what-if” load calculator (kW/kVA vs derating and efficiency) and an exportable commissioning report template.

VBA logigram and algorigram for access-layer design and oversubscription

You’ve got a rich access-layer design brief: Wi-Fi 6 mGig ports, PoE power tiers, MEC uplinks, and stack scaling. Below is a compact Excel VBA framework that turns this into a navigable logigram (topology and inventory) plus an algorigram (rules for oversubscription, PoE, resiliency).

Workbook schema

Create these sheets with exact headers.

Switches

- Columns: SwitchID, InStack, Model, PortsTotal, Ports_mGigCapable, mGigCap_SpeedMaxGbps, Ports_Gigabit, UplinkPorts_Total, UplinkPorts_Active, UplinkSpeedGbps, MEC_Enabled, PoE_Budget_W
- Example:
    - SW1, Yes, C9300-48, 48, 12, 10, 36, 4, 2, 10, Yes, 1440
    - SW2, Yes, C9300-48, 48, 12, 10, 36, 4, 2, 10, Yes, 1440
    - SW3, Yes, C9300-48, 48, 12, 10, 36, 4, 0, 10, No, 1440
    - SW4, Yes, C9300-48, 48, 12, 10, 36, 4, 0, 10, No, 1440

Loads

- Columns: SwitchID, WiFi6_AP_Count, AP_LinkGbps, Endpoints_1G_Count, Endpoints_1G_UtilizationPct, mGig_UsedPorts, mGig_OperGbps, UnusedPorts
- Example:
    - SW1, 8, 5, 32, 60, 0, 0, 8
    - SW2, 8, 5, 32, 60, 0, 0, 8
    - SW3, 0, 0, 36, 40, 0, 0, 12

- SW4, 0, 0, 36, 40, 0, 0, 12

StackPlan

- Columns: StackID, MembersCSV, ActiveUplinks_Total, UplinkSpeedGbps, MEC_Enabled, DesignTarget_Oversub_Max
- Example:
  - STK1, SW1,SW2, 4, 10, Yes, 4.0
  - STK2, SW3,SW4, 2, 10, Yes, 8.0

PoEProfiles

- Columns: DeviceType, Count, PerDevice_W
- Example:
  - AP_WiFi6, 8, 30
  - IP_Phone, 32, 9
  - Camera, 4, 13

What this engine does

- Computes worst-case and realistic oversubscription per switch and per stack.
- Accounts for mGig capable vs operating speeds (e.g., APs at 5 Gbps).
- Aggregates MEC uplinks into total uplink bandwidth.
- Checks PoE budget against attached devices.
- Builds a TreeView logigram and a ListView of findings.

Class: cNode

```
' Class Module: cNode
Option Explicit
Public ID As String
Public ParentID As String
Public Title As String
Public Kind As String          ' Switch | Stack | Calc | Finding
```

```
Public Meta As Scripting.Dictionary
Private Sub Class_Initialize()
    Set Meta = New Scripting.Dictionary
End Sub
' Module: mAccess
Option Explicit

' References required:
' - Microsoft Scripting Runtime
' - Microsoft Forms 2.0
' - Microsoft Windows Common Controls 6.0

Public Nodes As Scripting.Dictionary
Public ParentMap As Scripting.Dictionary

Public Sub Build()
    Set Nodes = New Scripting.Dictionary
    Set ParentMap = New Scripting.Dictionary

    BuildSwitches
    BuildStacks
End Sub

' -------- Switch-level build --------
Private Sub BuildSwitches()
    Ensure "ROOT", "", "Access-layer design", "Calc", Nothing
    Ensure "SW_ROOT", "ROOT", "Switches", "Calc", Nothing

    Dim wsS As Worksheet, wsL As Worksheet
    Set wsS = ThisWorkbook.Worksheets("Switches")
    Set wsL = ThisWorkbook.Worksheets("Loads")

    Dim lastS&, r&, sid$, rowL&, uplinksActive&, uplinkSpd#, mec As Boolean
    Dim portsTotal&, portsMGCap&, ports1G&, mgCapMax#, poeBudget#

    lastS = wsS.Cells(wsS.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastS
        sid = CStr(wsS.Cells(r, 1).Value2)
        portsTotal = CLng(wsS.Cells(r, 4).Value2)
        portsMGCap = CLng(wsS.Cells(r, 5).Value2)
        mgCapMax = CDbl(wsS.Cells(r, 6).Value2)
        ports1G = CLng(wsS.Cells(r, 7).Value2)
        uplinksActive = CLng(wsS.Cells(r, 9).Value2)
        uplinkSpd = CDbl(wsS.Cells(r, 10).Value2)
        mec = UCase$(CStr(wsS.Cells(r, 11).Value2)) = "YES"
        poeBudget = CDbl(Nz(wsS.Cells(r, 12).Value2, 0))

        ' Load row for this switch
        rowL = FindRow(wsL, 1, sid)
        Dim apCnt&, apGb#, epCnt&, epUtil#, mgUsed&, mgOperGb#, unused&
        If rowL > 0 Then
            apCnt = CLng(Nz(wsL.Cells(rowL, 2).Value2, 0))
            apGb = CDbl(Nz(wsL.Cells(rowL, 3).Value2, 0))
            epCnt = CLng(Nz(wsL.Cells(rowL, 4).Value2, 0))
            epUtil = CDbl(Nz(wsL.Cells(rowL, 5).Value2, 60))
```

```
            mgUsed = CLng(Nz(wsL.Cells(rowL, 6).Value2, 0))
            mgOperGb = CDbl(Nz(wsL.Cells(rowL, 7).Value2, 0))
            unused = CLng(Nz(wsL.Cells(rowL, 8).Value2, 0))
        End If

        Dim uplinkBW#:
        uplinkBW = uplinksActive * uplinkSpd

        ' Worst-case: assume all mGig-capable at their max, rest at 1G
        Dim accessWorst#:
        accessWorst = portsMGCap * mgCapMax + ports1G * 1#

        ' Realistic: Wi-Fi6 APs at apGb, remaining endpoints at 1G with
utilization
        Dim epReal#:
        epReal = epCnt * 1# * (epUtil / 100#)
        Dim mgReal#:
        mgReal = apCnt * apGb
        ' if explicit mGig used/oper provided, add them (other than APs)
        If mgUsed > 0 And mgOperGb > 0 Then mgReal = mgReal + (mgUsed *
mgOperGb)

        Dim accessReal#:
        accessReal = mgReal + epReal

        Dim overWorst#, overReal#:
        overWorst = SafeDiv(accessWorst, uplinkBW)
        overReal = SafeDiv(accessReal, uplinkBW)

        ' Findings thresholds
        Dim verdict$, note$
        verdict = OversubVerdict(overReal, 4#) ' default 4:1 target
        note = "Worst=" & Format(overWorst, "0.0") & ":1, Real=" &
Format(overReal, "0.0") & ":1, Uplinks=" & uplinksActive & "x" & uplinkSpd &
" (MEC=" & IIf(mec, "Yes", "No") & ")"

        Dim meta As Scripting.Dictionary: Set meta = New Scripting.Dictionary
        meta("PortsTotal") = portsTotal
        meta("mGigCapable") = portsMGCap & " @" & mgCapMax & "G"
        meta("GigabitPorts") = ports1G
        meta("APs@Gbps") = apCnt & " @" & apGb & "G"
        meta("Endpoints_1G") = epCnt & " @" & epUtil & "% util"
        meta("Access_Worst_Gbps") = Format(accessWorst, "0.0")
        meta("Access_Real_Gbps") = Format(accessReal, "0.0")
        meta("Uplink_Gbps") = Format(uplinkBW, "0.0")
        meta("Oversub_Worst") = Format(overWorst, "0.0") & ":1"
        meta("Oversub_Real") = Format(overReal, "0.0") & ":1"
        meta("Verdict") = verdict
        meta("Note") = note

        Ensure "SW_" & sid, "SW_ROOT", sid, "Switch", meta

        ' Optional PoE check
        Dim poeMeta As Scripting.Dictionary
        Set poeMeta = PoEBudgetCheck(sid, poeBudget)
```

```
        If Not poeMeta Is Nothing Then
            Ensure "SW_" & sid & "_POE", "SW_" & sid, "PoE budget check",
"Finding", poeMeta
        End If
    Next r
End Sub

' -------- Stack-level build --------
Private Sub BuildStacks()
    Ensure "STK_ROOT", "ROOT", "Stacks", "Calc", Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("StackPlan")
    Dim last&, r&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row

    For r = 2 To last
        Dim stk$, members$, target#, upl#, uplSpd#, mec As Boolean
        stk = CStr(ws.Cells(r, 1).Value2)
        members = CStr(ws.Cells(r, 2).Value2)
        upl = CLng(Nz(ws.Cells(r, 3).Value2, 0))
        uplSpd = CDbl(Nz(ws.Cells(r, 4).Value2, 10))
        mec = UCase$(CStr(ws.Cells(r, 5).Value2)) = "YES"
        target = CDbl(Nz(ws.Cells(r, 6).Value2, 4#))

        Dim arr() As String: arr = Split(members, ",")
        Dim i&, accessWorst#, accessReal#, uplinkBW#
        uplinkBW = upl * uplSpd

        For i = LBound(arr) To UBound(arr)
            Dim sid$: sid = Trim$(arr(i))
            Dim swMeta As Scripting.Dictionary
            Set swMeta = GetNodeMeta("SW_" & sid)
            If Not swMeta Is Nothing Then
                accessWorst = accessWorst + Val(swMeta("Access_Worst_Gbps"))
                accessReal = accessReal + Val(swMeta("Access_Real_Gbps"))
            End If
        Next i

        Dim overWorst#, overReal#:
        overWorst = SafeDiv(accessWorst, uplinkBW)
        overReal = SafeDiv(accessReal, uplinkBW)

        Dim meta As New Scripting.Dictionary
        meta("Members") = members
        meta("Access_Worst_Gbps") = Format(accessWorst, "0.0")
        meta("Access_Real_Gbps") = Format(accessReal, "0.0")
        meta("Uplink_Gbps") = Format(uplinkBW, "0.0")
        meta("Oversub_Worst") = Format(overWorst, "0.0") & ":1"
        meta("Oversub_Real") = Format(overReal, "0.0") & ":1"
        meta("Target_Max") = Format(target, "0.0") & ":1"
        meta("Verdict") = OversubVerdict(overReal, target)
        meta("MEC") = IIf(mec, "Yes", "No")

        Ensure "STK_" & stk, "STK_ROOT", stk, "Stack", meta
    Next r
End Sub
```

```
' -------- Helpers --------
Private Function OversubVerdict(ByVal over As Double, ByVal target As Double)
As String
    If over <= target Then
        OversubVerdict = "OK"
    ElseIf over <= target * 1.5 Then
        OversubVerdict = "Watch"
    Else
        OversubVerdict = "Hot"
    End If
End Function

Private Function PoEBudgetCheck(switchID As String, poeBudgetW As Double) As
Scripting.Dictionary
    On Error GoTo done
    If poeBudgetW <= 0 Then Exit Function
    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("PoEProfiles")
    Dim last&, r&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row

    Dim totalW#, details$
    For r = 2 To last
        Dim type$, cnt&, perW#
        type = CStr(ws.Cells(r, 1).Value2)
        cnt = CLng(Nz(ws.Cells(r, 2).Value2, 0))
        perW = CDbl(Nz(ws.Cells(r, 3).Value2, 0))
        totalW = totalW + cnt * perW
        If cnt > 0 Then details = details & type & "=" & cnt & "@" & perW &
"W; "
    Next r

    Dim d As New Scripting.Dictionary
    d("Budget_W") = Format(poeBudgetW, "0")
    d("Required_W") = Format(totalW, "0")
    d("Utilization") = IIf(poeBudgetW > 0, Format(100# * totalW / poeBudgetW,
"0") & "%", "n/a")
    d("Within_Budget") = IIf(totalW <= poeBudgetW, "Yes", "No")
    d("Devices") = details
    Set PoEBudgetCheck = d
done:
End Function

Private Function GetNodeMeta(nodeID As String) As Scripting.Dictionary
    If Nodes.Exists(nodeID) Then Set GetNodeMeta = Nodes(nodeID).Meta
End Function

Private Function FindRow(ws As Worksheet, colIndex As Long, key As String) As
Long
    Dim last&: last = ws.Cells(ws.Rows.Count, colIndex).End(xlUp).Row
    Dim r&
    For r = 2 To last
        If CStr(ws.Cells(r, colIndex).Value2) = key Then FindRow = r: Exit
Function
    Next r
End Function
```

```
Private Function Nz(v As Variant, Optional def As Variant) As Variant
    If IsError(v) Or IsEmpty(v) Or v = "" Then
        Nz = def
    Else
        Nz = v
    End If
End Function

Private Function SafeDiv(n#, d#) As Double
    If d = 0 Then SafeDiv = 0 Else SafeDiv = n / d
End Function

Public Sub Ensure(id$, parent$, title$, kind$, meta As Scripting.Dictionary)
    If Nodes Is Nothing Then Set Nodes = New Scripting.Dictionary
    If Not Nodes.Exists(id) Then
        Dim n As cNode: Set n = New cNode
        n.ID = id: n.ParentID = parent: n.Title = title: n.Kind = kind
        If Not meta Is Nothing Then
            Dim k: For Each k In meta.Keys: n.Meta(k) = meta(k): Next k
        End If
        Nodes(id) = n
        If Len(parent) > 0 Then AddChild parent, id
    End If
End Sub

Private Sub AddChild(parent$, child$)
    If ParentMap Is Nothing Then Set ParentMap = New Scripting.Dictionary
    If Not ParentMap.Exists(parent) Then
        Dim c As New Collection: Set ParentMap(parent) = c
    End If
    ParentMap(parent).Add child
End Sub
```

UserForm: frmAccess

- Controls:
  - TreeView: tvNav
  - ListView: lvMeta (View=Report; columns Key, Value)
  - Label: lblSummary

```
' UserForm: frmAccess
Option Explicit

Private Sub UserForm_Initialize()
    On Error Resume Next
    lvMeta.ColumnHeaders.Clear
    lvMeta.ColumnHeaders.Add , , "Key", 180
    lvMeta.ColumnHeaders.Add , , "Value", 300
    On Error GoTo 0
```

```
•
•       mAccess.Build
•       BuildTree
•       lblSummary.Caption = CStr(mAccess.Nodes.Count) & " nodes"
•   End Sub
•
•   Private Sub BuildTree()
•       tvNav.Nodes.Clear
•       Dim k
•       For Each k In mAccess.Nodes.Keys
•           Dim n As cNode: Set n = mAccess.Nodes(k)
•           If Len(n.ParentID) = 0 Then
•               tvNav.Nodes.Add , , n.ID, Prefix(n.Kind) & n.Title
•               AddChildren n.ID
•           End If
•       Next k
•       tvNav.ExpandAll
•   End Sub
•
•   Private Sub AddChildren(parent$)
•       If Not mAccess.ParentMap.Exists(parent) Then Exit Sub
•       Dim ch As Collection: Set ch = mAccess.ParentMap(parent)
•       Dim i&
•       For i = 1 To ch.Count
•           Dim cid$: cid = ch(i)
•           Dim n As cNode: Set n = mAccess.Nodes(cid)
•           tvNav.Nodes.Add n.ParentID, tvwChild, n.ID, Prefix(n.Kind) &
    n.Title
•           AddChildren n.ID
•       Next i
•   End Sub
•
•   Private Function Prefix(kind$) As String
•       Select Case kind
•           Case "Switch": Prefix = "[SW] "
•           Case "Stack": Prefix = "[STK] "
•           Case "Calc": Prefix = "[CALC] "
•           Case "Finding": Prefix = "• "
•           Case Else: Prefix = ""
•       End Select
•   End Function
•
•   Private Sub tvNav_NodeClick(ByVal Node As MSComctlLib.Node)
•       ShowMeta Node.Key
•   End Sub
•
•   Private Sub ShowMeta(nodeID$)
•       Dim n As cNode: Set n = mAccess.Nodes(nodeID)
•       lvMeta.ListItems.Clear
•       AddRow "Title", n.Title
•       AddRow "Kind", n.Kind
•       Dim k
•       For Each k In n.Meta.Keys
•           AddRow CStr(k), CStr(n.Meta(k))
•       Next k
```

```
End Sub

Private Sub AddRow(k$, v$)
    Dim it As ListItem
    Set it = lvMeta.ListItems.Add(, , k)
    it.SubItems(1) = v
End Sub
```

Notes and extensions

- mGig realism: APs capped at 5 Gbps by current 9100 series; you can raise as models evolve.
- Uplinks: MEC assumed aggregates bandwidth; if not MEC, adjust aggregation policy.
- Target oversub: default 4:1 at access; tune per policy per stack (DesignTarget_Oversub_Max).
- PoE: tie PoEProfiles per switch or stack as needed; add per-switch device mix if different.

Want me to wire this to your exact workbook and add an exportable “design compliance” report (oversub, PoE, resiliency) for your stack plan?

VBA logigram and algorigram for switch stack design, oversubscription, and uplink planning

You’ve got everything to build a living design model: stack behavior, mGig/PoE realities, MEC uplinks, failure-mode headroom, and fiber uplink migration (10G→25G). Below is a compact VBA framework that turns this into a navigable logigram (inventory, topology, optics) and an algorigram (rules for oversubscription, failure scenarios, and optics distance compatibility).

Workbook schema

Create these sheets with the exact headers and example rows.

Switches

- Columns: SwitchID, StackID, Role, PortsTotal, Ports_mGigCapable, mGigCap_SpeedMaxGbps, Ports_Gigabit, UplinkPorts_Total, UplinkPorts_Active, UplinkSpeedGbps, MEC_Enabled, PoE_Budget_W, CarriesUplinks

- Example:
    - SW1, STK1, Member, 48, 12, 10, 36, 4, 2, 10, Yes, 1440, Yes
    - SW2, STK1, Member, 48, 12, 10, 36, 4, 2, 10, Yes, 1440, Yes
    - SW3, STK1, Active, 48, 12, 10, 36, 4, 0, 10, No, 1440, No
    - SW4, STK1, Standby, 48, 12, 10, 36, 4, 0, 10, No, 1440, No

Loads

- Columns: SwitchID, AP_Count, AP_OperGbps, Endpoints_1G_Count, Endpoints_1G_UtilPct, mGig_NonAP_Count, mGig_NonAP_OperGbps, UnusedPorts
- Example:
    - SW1, 8, 5, 32, 60, 0, 0, 8
    - SW2, 8, 5, 32, 60, 0, 0, 8
    - SW3, 0, 0, 36, 40, 0, 0, 12
    - SW4, 0, 0, 36, 40, 0, 0, 12

StackPlan

- Columns: StackID, DesignTarget_Oversub_Max, EtherChannel_MaxLinks, FailureMode_Check, Notes
- Example:
    - STK1, 4.0, 8, Yes, MEC across uplink-carrying members; balance APs across stack

Optics

- Columns: Module, SpeedGbps, Mode, FiberType, MaxDistance_m, DualRate
- Examples:
    - 10G-SR, 10, MMF, OM3, 300, No

- 10G-SR, 10, MMF, OM4, 400, No
- 10/25G-CSR, 10, MMF, OM3, 300, Yes
- 10/25G-CSR, 10, MMF, OM4, 400, Yes
- 10/25G-CSR, 25, MMF, OM3, 300, Yes
- 10/25G-CSR, 25, MMF, OM4, 400, Yes
- 25G-LR, 25, SMF, SMF, 10000, No

FiberPlant

- Columns: LinkID, From, To, FiberType, Distance_m, PatchLosses_dB
- Example:
  - L1, Dist-1, STK1, OM3, 220, 1.0

What the engine computes

- Per-switch worst-case vs realistic access bandwidth.
- Per-stack oversubscription for normal operation and failure modes:
  - Loss of a member carrying uplinks.
  - Recalculate remaining active uplinks and remaining access load.
- MEC and EtherChannel constraints (max 8×10G = 80 Gbps).
- Optics distance compatibility for 10G and 25G, with dual-rate migration hints.
- Role placement best practice: Active/Standby on non-uplink members.

Class model

```
' Class Module: cNode
Option Explicit
Public ID As String
Public ParentID As String
Public Title As String
Public Kind As String          ' Switch | Stack | Optic | Link | Finding |
Calc
```

```
Public Meta As Scripting.Dictionary
Private Sub Class_Initialize()
    Set Meta = New Scripting.Dictionary
End Sub
```

Core engine

To model this multilayer campus LAN architecture in VBA—complete with logigram (topology, platforms, uplinks) and algorigram (oversubscription, fiber compatibility, security posture)—we'll build a modular framework that reflects:

- **Core, Distribution, and Access layers**
- **Oversubscription ratios between layers**
- **Uplink speed planning (40G vs 100G)**
- **Fiber type compatibility (SMF/MMF)**
- **Security features like DHCP Snooping**
- **StackWise and modular chassis behavior**

Workbook Schema

Create these sheets with exact headers:

Topology

Node ID	Layer	Role	Platform	Ports_Total	Uplink_Speed_Gbps	Uplink_Count	Stackwise	Modular	SMF_Ready
CORE1	Core	Primary	C9600	48	100	2	No	Yes	Yes
DIST1	Distribution	Aggregator	C9500	48	40	2	No	No	Yes
DIST2	Distribution	Aggregator	C9500	48	40	2	No	No	Yes
ACCESS1	Access	StackMember	C9300	48	10	4	Yes	No	No

UplinkMatrix

FromNode	ToNode	LinkSpeed_Gbps	LinkCount	FiberType	Distance_m
ACCESS1	DIST1	10	4	OM3	120
DIST1	CORE1	40	2	SMF	300

FromNode	ToNode	LinkSpeed_Gbps	LinkCount	FiberType	Distance_m
DIST2	CORE1	40	2	SMF	300

SecurityFeatures

NodeID	DHCP_Snooping	PortSecurity	Umbrella_Enabled
ACCESS1	Yes	Yes	Yes
DIST1	Yes	Yes	No
CORE1	No	No	No

□ Logigram + Algorigram VBA Engine

Class: cNode

```
' Class Module: cNode
Option Explicit
Public ID As String
Public ParentID As String
Public Title As String
Public Kind As String
Public Meta As Scripting.Dictionary
Private Sub Class_Initialize()
    Set Meta = New Scripting.Dictionary
End Sub
ption Explicit

Public Nodes As Scripting.Dictionary
Public ParentMap As Scripting.Dictionary

Public Sub Build()
    Set Nodes = New Scripting.Dictionary
    Set ParentMap = New Scripting.Dictionary

    BuildTopology
    BuildUplinks
    BuildSecurity
End Sub

Private Sub BuildTopology()
    Ensure "ROOT", "", "Campus LAN Architecture", "Layer", Nothing
    Ensure "TOPO", "ROOT", "Topology", "Layer", Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Sheets("Topology")
    Dim r&, lastRow&: lastRow = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row

    For r = 2 To lastRow
```

```
        Dim id$, layer$, role$, plat$, ports&, uplSpd#, uplCnt&, stack$,
mod$, smf$
        id = ws.Cells(r, 1).Value2
        layer = ws.Cells(r, 2).Value2
        role = ws.Cells(r, 3).Value2
        plat = ws.Cells(r, 4).Value2
        ports = ws.Cells(r, 5).Value2
        uplSpd = ws.Cells(r, 6).Value2
        uplCnt = ws.Cells(r, 7).Value2
        stack = ws.Cells(r, 8).Value2
        mod = ws.Cells(r, 9).Value2
        smf = ws.Cells(r, 10).Value2

        Dim meta As New Scripting.Dictionary
        meta("Layer") = layer
        meta("Role") = role
        meta("Platform") = plat
        meta("Ports") = ports
        meta("UplinkSpeed") = uplSpd
        meta("UplinkCount") = uplCnt
        meta("Stackwise") = stack
        meta("Modular") = mod
        meta("SMF_Ready") = smf

        Ensure "NODE_" & id, "TOPO", id, "Node", meta
    Next r
End Sub

Private Sub BuildUplinks()
    Ensure "UPLINKS", "ROOT", "Uplink Matrix", "Link", Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Sheets("UplinkMatrix")
    Dim r&, lastRow&: lastRow = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row

    For r = 2 To lastRow
        Dim from$, to$, spd#, cnt&, fiber$, dist&
        from = ws.Cells(r, 1).Value2
        to = ws.Cells(r, 2).Value2
        spd = ws.Cells(r, 3).Value2
        cnt = ws.Cells(r, 4).Value2
        fiber = ws.Cells(r, 5).Value2
        dist = ws.Cells(r, 6).Value2

        Dim meta As New Scripting.Dictionary
        meta("From") = from
        meta("To") = to
        meta("SpeedGbps") = spd
        meta("LinkCount") = cnt
        meta("FiberType") = fiber
        meta("Distance_m") = dist
        meta("OversubRatio") = ComputeOversub(from, to, spd, cnt)

        Ensure "LINK_" & from & "_" & to, "UPLINKS", from & " → " & to,
"Link", meta
    Next r
```

```
End Sub

Private Function ComputeOversub(from$, to$, spd#, cnt&) As String
    Dim accessBW#, coreBW#
    accessBW = Val(NodeMeta("NODE_" & from, "UplinkSpeed")) *
Val(NodeMeta("NODE_" & from, "UplinkCount"))
    coreBW = spd * cnt
    If coreBW = 0 Then ComputeOversub = "∞" Else ComputeOversub =
Format(accessBW / coreBW, "0.0") & ":1"
End Function

Private Sub BuildSecurity()
    Ensure "SECURITY", "ROOT", "Security Features", "Security", Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Sheets("SecurityFeatures")
    Dim r&, lastRow&: lastRow = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row

    For r = 2 To lastRow
        Dim id$, dhcp$, portsec$, umb$
        id = ws.Cells(r, 1).Value2
        dhcp = ws.Cells(r, 2).Value2
        portsec = ws.Cells(r, 3).Value2
        umb = ws.Cells(r, 4).Value2

        Dim meta As New Scripting.Dictionary
        meta("DHCP_Snooping") = dhcp
        meta("PortSecurity") = portsec
        meta("Umbrella") = umb

        Ensure "SEC_" & id, "SECURITY", id, "Security", meta
    Next r
End Sub

Private Function NodeMeta(id$, key$) As String
    If Nodes.Exists(id) Then
        If Nodes(id).Meta.Exists(key) Then NodeMeta = Nodes(id).Meta(key)
    End If
End Function

Public Sub Ensure(id$, parent$, title$, kind$, meta As Scripting.Dictionary)
    If Nodes Is Nothing Then Set Nodes = New Scripting.Dictionary
    If Not Nodes.Exists(id) Then
        Dim n As cNode: Set n = New cNode
        n.ID = id: n.ParentID = parent: n.Title = title: n.Kind = kind
        If Not meta Is Nothing Then
            Dim k: For Each k In meta.Keys: n.Meta(k) = meta(k): Next k
        End If
        Nodes(id) = n
        If Len(parent) > 0 Then AddChild parent, id
    End If
End Sub

Private Sub AddChild(parent$, child$)
    If ParentMap Is Nothing Then Set ParentMap = New Scripting.Dictionary
    If Not ParentMap.Exists(parent) Then
```

```
        Dim c As New Collection: Set ParentMap(parent) = c
    End If
    ParentMap(parent).Add child
End Sub
```

□ Navigation UserForm: `frmCampusLAN`

```
rivate Sub UserForm_Initialize()
    lvMeta.ColumnHeaders.Clear
    lvMeta.ColumnHeaders.Add , , "Key", 180
    lvMeta.ColumnHeaders.Add , , "Value", 300

    mCampusLAN.Build
    BuildTree
    lblSummary.Caption = CStr(mCampusLAN.Nodes.Count) & " nodes loaded"
End Sub

Private Sub BuildTree()
    tvNav.Nodes.Clear
    Dim k
    For Each k In mCampusLAN.Nodes.Keys
        Dim n As cNode: Set n = mCampusLAN.Nodes(k)
        If Len(n.ParentID) = 0 Then
            tvNav.Nodes.Add , , n.ID, "[" & n
```

VBA logigram and algorigram for high availability and wireless controller design

You've got two rich domains to model:

- High availability features across Catalyst 9200/9300/9400/9500/9600.
- Wireless stack design around Catalyst 9800 controllers and 9100 APs.

Below is a compact, production-ready VBA framework that:

- Builds a logigram of HA capabilities and WLAN platforms.
- Runs algorigram decisions to recommend controller SKUs based on AP/client targets and deployment mode.
- Flags HA gaps (ISSU, GIR, power redundancy) and stacking design choices.

Workbook schema

Create these sheets with exact headers.

HA_Features

- Columns: Platform, SwitchStacking, SupRedundancy, NSF_SSO, EtherChannel, ISSU, SMUs, GIR, PowerRedundancy

- Rows (examples):

    - Cisco Catalyst 9200 Series | StackWise-160/80 with Active/Standby | — | Yes | Cross-Stack EtherChannel | No | Yes | No | Up to 2 hot-swappable PSUs (PoE=Combined, Non-PoE=1:1)

    - Cisco Catalyst 9300 Series | StackWise-480/360 with Active/Standby | — | Yes | Cross-Stack EtherChannel | No (FSU/Ext FSU) | Yes | Yes | StackPower up to 4 (XPS up to 8)

    - Cisco Catalyst 9400 Series | — | Single chassis 1:1 or cross chassis StackWise Virtual | Yes | MEC with SV | Yes | Yes | Yes | Hot-swappable PSUs in N+N or N+1

    - Cisco Catalyst 9500 Series | — | Cross chassis StackWise Virtual | Yes | MEC with SV | Yes | Yes | Yes | Dual 1+1 PSUs

    - Cisco Catalyst 9600 Series | — | Single chassis 1:1 or cross chassis StackWise Virtual | Yes | MEC with SV | Yes | Yes | Yes | 4 PSUs (Combined or N+1)

WLAN_Controllers

- Columns: Platform, DeploymentMode, Topology, MaxAPs, MaxClients, ThroughputGbps, Notes

- Rows (examples):

    - 9800-80 | Centralized;FlexConnect;SD-Access | Large Campus | 6000 | 64000 | 80 | —

    - 9800-40 | Centralized;FlexConnect;SD-Access | Medium Campus | 2000 | 32000 | 40 | —

    - 9800-L | Centralized;FlexConnect;SD-Access | Small/Remote | 250 | 5000 | 5 | —

    - 9800-L Performance | Centralized;FlexConnect;SD-Access | Small/Remote | 500 | 10000 | 9 | Perf license

- 9800 Embedded on C9000 | SD-Access | Small Distributed | 200 | 4000 | — | Local switching
- 9800 EWC on 9100 AP | Local Switching | Small Remote | 100 | 2000 | — | Local switching
- 9800-CL Public Cloud | FlexConnect (Local) | Virtual Small Remote | 1000/3000/6000 | 10000/32000/64000 | — | Local switching
- 9800-CL Private Cloud | Centralized;FlexConnect;SD-Access | Virtual Small/Med/Large | 1000/3000/6000 | 10000/32000/64000 | 2.1 (central) | IOS-XE ≥17.1

WLAN_Design

- Columns: SiteID, AP_Count, Client_Count, DeploymentPref, TopologyPref, CentralSwitching, HA_Required, AlwaysOn_Upgrade, Notes
- Example:
  - Campus_A | 1800 | 20000 | Centralized | Large Campus | Yes | Yes | Seamless updates

Class model

vba

```
' Class Module: cNode
Option Explicit
Public ID As String
Public ParentID As String
Public Title As String
Public Kind As String          ' HA | WLAN | Finding | Recommendation
Public Meta As Scripting.Dictionary
Private Sub Class_Initialize()
    Set Meta = New Scripting.Dictionary
End Sub
```

Core engine: logigram + algorigram

vba

```
' Module: mCampusHAWireless
Option Explicit

' References required:
```

```
' - Microsoft Scripting Runtime
' - Microsoft Forms 2.0
' - Microsoft Windows Common Controls 6.0 (SP6)

Public Nodes As Scripting.Dictionary
Public ParentMap As Scripting.Dictionary

Public Sub Build()
    Set Nodes = New Scripting.Dictionary
    Set ParentMap = New Scripting.Dictionary
    BuildHA
    BuildWLAN
    EvaluateDesigns
End Sub

' ---------- High Availability features ----------
Private Sub BuildHA()
    Ensure "ROOT", "", "Campus high availability and wireless design", "HA",
Nothing
    Ensure "HA_ROOT", "ROOT", "High availability matrix", "HA", Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("HA_Features")
    Dim r&, lastRow&: lastRow = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row

    For r = 2 To lastRow
        Dim plat$, stack$, sup$, nsf$, ec$, issu$, smu$, gir$, pwr$
        plat = CStr(ws.Cells(r, 1).Value2)
        stack = CStr(ws.Cells(r, 2).Value2)
        sup = CStr(ws.Cells(r, 3).Value2)
        nsf = CStr(ws.Cells(r, 4).Value2)
        ec = CStr(ws.Cells(r, 5).Value2)
        issu = CStr(ws.Cells(r, 6).Value2)
        smu = CStr(ws.Cells(r, 7).Value2)
        gir = CStr(ws.Cells(r, 8).Value2)
        pwr = CStr(ws.Cells(r, 9).Value2)

        Dim meta As New Scripting.Dictionary
        meta("Stacking") = stack
        meta("SupervisorRedundancy") = sup
        meta("NSF/SSO") = nsf
        meta("EtherChannel") = ec
        meta("ISSU") = issu
        meta("SMUs") = smu
        meta("GIR") = gir
        meta("Power") = pwr
        meta("HA_Score") = HAScore(nsf, issu, gir, pwr)

        Ensure "HA_" & Normalize(plat), "HA_ROOT", plat, "HA", meta
    Next r
End Sub

Private Function HAScore(nsf$, issu$, gir$, pwr$) As String
    Dim score As Long: score = 0
    If Yes(nsf) Then score = score + 3
    If Yes(issu) Then score = score + 3
```

```
    If Yes(gir) Then score = score + 2
    If InStr(1, UCase$(pwr$), "N+1") > 0 Or InStr(1, UCase$(pwr$), "N+N") > 0
Then score = score + 2 Else score = score + 1
    HAScore = CStr(score) & "/10"
End Function

' ---------- WLAN platforms ----------
Private Sub BuildWLAN()
    Ensure "WLAN_ROOT", "ROOT", "Wireless controllers", "WLAN", Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("WLAN_Controllers")
    Dim r&, lastRow&: lastRow = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastRow
        Dim plat$, dep$, topo$, maxAP&, maxCli&, thp$, notes$
        plat = CStr(ws.Cells(r, 1).Value2)
        dep = CStr(ws.Cells(r, 2).Value2)
        topo = CStr(ws.Cells(r, 3).Value2)
        maxAP = CLng(Nz(ws.Cells(r, 4).Value2, 0))
        maxCli = CLng(Nz(ws.Cells(r, 5).Value2, 0))
        thp = CStr(ws.Cells(r, 6).Value2)
        notes = CStr(ws.Cells(r, 7).Value2)

        Dim meta As New Scripting.Dictionary
        meta("DeploymentMode") = dep
        meta("Topology") = topo
        meta("MaxAPs") = maxAP
        meta("MaxClients") = maxCli
        meta("ThroughputGbps") = thp
        If Len(notes) > 0 Then meta("Notes") = notes

        Ensure "WLC_" & Normalize(plat), "WLAN_ROOT", plat, "WLAN", meta
    Next r
End Sub

' ---------- Design evaluator (algorigram) ----------
Private Sub EvaluateDesigns()
    Ensure "DESIGN_ROOT", "ROOT", "Design recommendations", "Recommendation",
Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("WLAN_Design")
    Dim r&, lastRow&: lastRow = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    If lastRow < 2 Then Exit Sub

    For r = 2 To lastRow
        Dim site$, ap&, cli&, depPref$, topoPref$, central$, haReq$,
alwaysOn$
        site = CStr(ws.Cells(r, 1).Value2)
        ap = CLng(Nz(ws.Cells(r, 2).Value2, 0))
        cli = CLng(Nz(ws.Cells(r, 3).Value2, 0))
        depPref = CStr(ws.Cells(r, 4).Value2)
        topoPref = CStr(ws.Cells(r, 5).Value2)
        central = CStr(ws.Cells(r, 6).Value2)
        haReq = CStr(ws.Cells(r, 7).Value2)
        alwaysOn = CStr(ws.Cells(r, 8).Value2)
```

```
        Dim pick As Scripting.Dictionary: Set pick = PickController(ap, cli,
depPref, topoPref, central, haReq)
        Dim meta As New Scripting.Dictionary
        meta("APs_Target") = ap
        meta("Clients_Target") = cli
        meta("Pref_Deployment") = depPref
        meta("Pref_Topology") = topoPref
        meta("CentralSwitching") = central
        meta("HA_Required") = haReq
        meta("AlwaysOn_Upgrade") = alwaysOn

        If Not pick Is Nothing Then
            Dim k
            For Each k In pick.Keys: meta(k) = pick(k): Next k
        Else
            meta("Recommendation") = "Review inputs; no matching controller"
        End If

        Ensure "DESIGN_" & Normalize(site), "DESIGN_ROOT", site,
"Recommendation", meta

        ' HA adjunct recommendation: distribution/core platform hint based on
HA requirements
        Dim haHint As String: haHint = HAHint(haReq, alwaysOn)
        AddFinding "DESIGN_" & Normalize(site) & "_HAHINT", "DESIGN_" &
Normalize(site), "HA platform hint", DictKV("Hint", haHint))
    Next r
End Sub

Private Function PickController(ap&, cli&, depPref$, topoPref$, central$,
haReq$) As Scripting.Dictionary
    Dim bestID$, bestHeadroom As Double: bestHeadroom = -1

    Dim k
    For Each k In Nodes.Keys
        If Left$(k, 4) = "WLC_" Then
            Dim n As cNode: Set n = Nodes(k)
            Dim dep As String: dep = UCase$(n.Meta("DeploymentMode"))
            Dim topo As String: topo = UCase$(n.Meta("Topology"))
            Dim capAP&, capCli&
            capAP = ValDef(n.Meta, "MaxAPs", 0)
            capCli = ValDef(n.Meta, "MaxClients", 0)

            ' Mode match
            If Len(depPref$) > 0 Then
                If InStr(1, dep, UCase$(depPref$)) = 0 Then GoTo NextWLC
            End If
            If Len(topoPref$) > 0 Then
                If InStr(1, topo, UCase$(topoPref$)) = 0 Then GoTo NextWLC
            End If
            If UCase$(central$) = "YES" Then
                ' Prefer platforms with explicit centralized throughput value
                If Not n.Meta.Exists("ThroughputGbps") Then GoTo NextWLC
            End If
```

```
            ' Capacity fit
            If capAP > 0 And capCli > 0 Then
                If ap <= capAP And cli <= capCli Then
                    Dim head As Double
                    head = (capAP - ap) / Application.Max(1, capAP) + (capCli
- cli) / Application.Max(1, capCli)
                    If head > bestHeadroom Then
                        bestHeadroom = head
                        bestID = k
                    End If
                End If
            ElseIf capAP > 0 And ap <= capAP Then
                If 0.1 > bestHeadroom Then bestHeadroom = 0.1: bestID = k
            End If
        End If
NextWLC:
    Next k

    If Len(bestID) = 0 Then Exit Function

    Dim pick As New Scripting.Dictionary
    Dim m As cNode: Set m = Nodes(bestID)
    pick("Recommendation") = Replace(bestID, "WLC_", "")
    pick("MaxAPs") = ValDef(m.Meta, "MaxAPs", 0)
    pick("MaxClients") = ValDef(m.Meta, "MaxClients", 0)
    If m.Meta.Exists("ThroughputGbps") Then pick("ThroughputGbps") =
m.Meta("ThroughputGbps")
    pick("DeploymentMode") = m.Meta("DeploymentMode")
    pick("Topology") = m.Meta("Topology")
    PickController = pick
End Function

Private Function HAHint(haReq$, alwaysOn$) As String
    If Yes(haReq$) And Yes(alwaysOn$) Then
        HAHint = "Favor 9400/9600 at dist/core for ISSU+GIR; 9300 stacks at
access with XPS/StackPower."
    ElseIf Yes(haReq$) Then
        HAHint = "9500 SV at distribution with MEC; 9300 StackWise at
access."
    Else
        HAHint = "9200/9300 at access; 9500 at distribution; right-size
core."
    End If
End Function

' ---------- Helpers ----------
Private Function Yes(v$) As Boolean
    Dim u$: u = UCase$(Trim$(v))
    Yes = (u = "YES" Or u = "Y" Or u = "TRUE")
End Function

Private Function DictKV(ParamArray kv()) As Scripting.Dictionary
    Dim d As New Scripting.Dictionary, i&
    For i = LBound(kv) To UBound(kv) Step 2
        d(CStr(kv(i))) = CStr(kv(i + 1))
```

```
    Next i
    Set DictKV = d
End Function

Private Function Nz(v As Variant, Optional def As Variant) As Variant
    If IsError(v) Or IsEmpty(v) Or v = "" Then Nz = def Else Nz = v
End Function

Private Function ValDef(meta As Scripting.Dictionary, key$, def#) As Double
    If meta.Exists(key) Then ValDef = Val(meta(key)) Else ValDef = def
End Function

Private Sub Ensure(id$, parent$, title$, kind$, meta As Scripting.Dictionary)
    If Nodes Is Nothing Then Set Nodes = New Scripting.Dictionary
    If Not Nodes.Exists(id) Then
        Dim n As cNode: Set n = New cNode
        n.ID = id: n.ParentID = parent: n.Title = title: n.Kind = kind
        If Not meta Is Nothing Then
            Dim k: For Each k In meta.Keys: n.Meta(k) = meta(k): Next k
        End If
        Nodes(id) = n
        If Len(parent) > 0 Then AddChild parent, id
    End If
End Sub

Private Sub AddChild(parent$, child$)
    If ParentMap Is Nothing Then Set ParentMap = New Scripting.Dictionary
    If Not ParentMap.Exists(parent) Then
        Dim c As New Collection: Set ParentMap(parent) = c
    End If
    ParentMap(parent).Add child
End Sub

Private Function Normalize(ByVal s As String) As String
    Dim t$: t = Trim$(s)
    t = Replace(t, " ", "_"): t = Replace(t, "/", "_"): t = Replace(t, "-",
"_")
    t = Replace(t, "(", "_"): t = Replace(t, ")", "_"): t = Replace(t, ".",
"_")
    Normalize = UCase$(t)
End Function
' UserForm: frmHAWLAN
Option Explicit

Private Sub UserForm_Initialize()
    lvMeta.ColumnHeaders.Clear
    lvMeta.ColumnHeaders.Add , , "Key", 180
    lvMeta.ColumnHeaders.Add , , "Value", 320

    mCampusHAWireless.Build
    BuildTree
    lblSummary.Caption = CStr(mCampusHAWireless.Nodes.Count) & " nodes"
End Sub

Private Sub BuildTree()
```

```
    tvNav.Nodes.Clear
    Dim k
    For Each k In mCampusHAWireless.Nodes.Keys
        Dim n As cNode: Set n = mCampusHAWireless.Nodes(k)
        If Len(n.ParentID) = 0 Then
            tvNav.Nodes.Add , , n.ID, Prefix(n.Kind) & n.Title
            AddChildren n.ID
        End If
    Next k
    tvNav.ExpandAll
End Sub

Private Sub AddChildren(parent$)
    If Not mCampusHAWireless.ParentMap.Exists(parent) Then Exit Sub
    Dim ch As Collection: Set ch = mCampusHAWireless.ParentMap(parent)
    Dim i&
    For i = 1 To ch.Count
        Dim cid$: cid = ch(i)
        Dim n As cNode: Set n = mCampusHAWireless.Nodes(cid)
        tvNav.Nodes.Add n.ParentID, tvwChild, n.ID, Prefix(n.Kind) & n.Title
        AddChildren n.ID
    Next i
End Sub

Private Function Prefix(kind$) As String
    Select Case kind
        Case "HA": Prefix = "[HA] "
        Case "WLAN": Prefix = "[WLC] "
        Case "Recommendation": Prefix = "• "
        Case "Finding": Prefix = "• "
        Case Else: Prefix = ""
    End Select
End Function

Private Sub tvNav_NodeClick(ByVal Node As MSComctlLib.Node)
    ShowMeta Node.Key
End Sub

Private Sub ShowMeta(nodeID$)
    Dim n As cNode: Set n = mCampusHAWireless.Nodes(nodeID)
    lvMeta.ListItems.Clear
    AddRow "Title", n.Title
    AddRow "Kind", n.Kind
    Dim k
    For Each k In n.Meta.Keys
        AddRow CStr(k), CStr(n.Meta(k))
    Next k
End Sub

Private Sub AddRow(k$, v$)
    Dim it As ListItem
    Set it = lvMeta.ListItems.Add(, , k)
    it.SubItems(1) = v
End Sub
```

What you get

- HA logigram of platforms with a simple 10-point HA score (NSF/SSO, ISSU, GIR, power).
- WLAN controller catalog mapped to deployment/topology, capacity, and throughput.
- Algorigram that selects the right 9800 platform for your AP/client demands and preferred mode, plus HA platform hints for distribution/core.
- Ready to export into commissioning or design review docs.

Want me to add an export sheet that prints a side-by-side of your site inputs vs the recommended controller and HA platform hints?

VBA logigram and algorigram for campus platform selection and wireless capabilities

You've got tiered platform guidance (Tables 7 & 8), rich Wi-Fi 6 AP capability sets, and acronyms galore. Below is a compact, production-ready VBA framework that:

- Builds a logigram of platforms for medium and high-density campus designs (core/distribution/access).
- Maps wireless AP capabilities (Wi-Fi 6, OFDMA, MU-MIMO, CleanAir, FRA, RF ASIC, mGig, radios).
- Runs an algorigram to recommend platforms (switches, controllers, AP series) by density and HA tier.
- Checks feature requirements vs. AP capability sets and flags gaps.
- Organizes a glossary for quick expansion of acronyms in the UI.

Workbook schema

Create these sheets with exact headers.

Platform_Tiers

- Columns: DensityLevel, Layer, Tier, Platform, Notes

- Rows (seed from Tables 7 & 8):
    - Medium, Distribution, Enterprise, Cisco Catalyst 9400 Series, Base foundation
    - Medium, Distribution, Advanced, Cisco Catalyst 9500 Series, Foundation+
    - Medium, Distribution, Mission, Cisco Catalyst 9600 Series, Best-in-class
    - Medium, Access, Enterprise, Cisco Catalyst 9200/9200-L Series, —
    - Medium, Access, Advanced, Cisco Catalyst 9300/9300-L Series, —
    - Medium, Access, Mission, Cisco Catalyst 9400 Series, —
    - Medium, WLC, Enterprise, Cisco Catalyst 9800-40 or 9800-CL, —
    - Medium, WLC, Advanced, 9800-40 HA SSO or N+1, —
    - Medium, WLC, Mission, 9800-40 HA SSO pair, —
    - Medium, AP, Enterprise, 9115AX or 9117AX, —
    - Medium, AP, Advanced, 9120AX, —
    - Medium, AP, Mission, 9130AX, —
    - High, Core, Enterprise, Cisco Catalyst 9500 Series, Lower-density fixed core
    - High, Core, Advanced, Cisco Catalyst 9600 Series, High-density modular
    - High, Core, Mission, Cisco Catalyst 9600 Series, Best-in-class
    - High, Distribution, Enterprise, Cisco Catalyst 9500 Series, —
    - High, Distribution, Advanced, Cisco Catalyst 9600 Series, —
    - High, Distribution, Mission, Cisco Catalyst 9600 Series, —

- High, Access, Enterprise, Cisco Catalyst 9300/9300-L Series, —
- High, Access, Advanced, Cisco Catalyst 9400 Series, —
- High, Access, Mission, Cisco Catalyst 9400 Series, —
- High, WLC, Enterprise, 9800-40/9800-CL, Centralized preferred
- High, WLC, Advanced, 9800-80 or 9800-40 HA SSO, —
- High, WLC, Mission, 9800-80 HA SSO, —
- High, AP, Enterprise, 9120AX, CleanAir/FRA
- High, AP, Advanced, 9130AX, 8x8 options
- High, AP, Mission, 9130AX, —

AP_Capabilities

- Columns: APSeries, CapabilitiesCSV, Radios, RF_ASIC, CleanAir, FRA, MU_MIMO, OFDMA, mGig, BLE_IoT
- Rows (examples, per your text):
  - 9115AX, WiFi6;MU-MIMO;OFDMA;BSS Coloring;TWT;Apple, 2.4(4x4),5(4x4) or (8x8), No, Yes, Limited, Yes, Yes, Yes, Yes
  - 9117AX, WiFi6;MU-MIMO;OFDMA;BSS Coloring;TWT;Apple, 2.4(4x4),5(8x8), No, Yes, Limited, Yes, Yes, Yes, Yes
  - 9120AX, WiFi6;MU-MIMO;OFDMA;BSS Coloring;TWT;Apple;Intelligent Capture;Container, 2.4(4x4),5(4x4), Yes, Yes, Yes, Yes, Yes, Yes, Yes
  - 9130AX, WiFi6 certified;MU-MIMO;OFDMA;BSS Coloring;TWT;Apple;Intelligent Capture;Container, 2.4(4x4),5(8x8 and 4x4), Yes, Yes, Yes, Yes, Yes, Yes, Yes

WLC_Profiles

- Columns: WLC, DeploymentModes, Topology, MaxAPs, MaxClients, ThroughputGbps, HAOptions
- Rows (subset):

- 9800-80, Centralized;FlexConnect;SD-Access, Large Campus, 6000, 64000, 80, HA SSO 1:1, N+1
- 9800-40, Centralized;FlexConnect;SD-Access, Medium Campus, 2000, 32000, 40, HA SSO 1:1, N+1
- 9800-L, Centralized;FlexConnect;SD-Access, Small/Remote, 250, 5000, 5, N+1
- 9800-CL, FlexConnect;Centralized;SD-Access, Virtual, 1000/3000/6000, 10000/32000/64000, 2.1 (central), Cloud

Sites

- Columns: SiteID, DensityLevel, HATier, AP_Count, Clients, WirelessMode, CentralizedPreferred, RequiredFeaturesCSV, Notes
- Example:
  - Campus_M1, Medium, Advanced, 120, 3500, Unified, Yes, RF_ASIC;CleanAir;FRA;mGig, —
  - Campus_H1, High, Mission, 1800, 25000, Unified, Yes, RF_ASIC;CleanAir;FRA;8x8, —

Glossary

- Columns: Term, Expansion
- Seed terms from your appendix (AAA, ACL, AP, CAPWAP, CleanAir, FRA, RF ASIC, etc.).

Class model

vba

```
' Class Module: cNode
Option Explicit
Public ID As String
Public ParentID As String
Public Title As String
Public Kind As String          ' Tier | AP | WLC | Site | Finding |
Recommendation | Glossary
Public Meta As Scripting.Dictionary
Private Sub Class_Initialize()
    Set Meta = New Scripting.Dictionary
End Sub
```

```
' Module: mCampusDesign
Option Explicit

' References:
' - Microsoft Scripting Runtime
' - Microsoft Forms 2.0
' - Microsoft Windows Common Controls 6.0 (TreeView/ListView)

Public Nodes As Scripting.Dictionary
Public ParentMap As Scripting.Dictionary

Public Sub Build()
    Set Nodes = New Scripting.Dictionary
    Set ParentMap = New Scripting.Dictionary

    BuildTiers
    BuildAPs
    BuildWLCs
    BuildGlossary
    EvaluateSites
End Sub

' --------- Platform tiers (Tables 7 & 8) ----------
Private Sub BuildTiers()
    Ensure "ROOT", "", "Campus design knowledge base", "Tier", Nothing
    Ensure "TIER_ROOT", "ROOT", "Platform tiers", "Tier", Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("Platform_Tiers")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row

    For r = 2 To last
        Dim dens$, layer$, tier$, plat$, notes$
        dens = CStr(ws.Cells(r, 1).Value2)
        layer = CStr(ws.Cells(r, 2).Value2)
        tier = CStr(ws.Cells(r, 3).Value2)
        plat = CStr(ws.Cells(r, 4).Value2)
        notes = CStr(ws.Cells(r, 5).Value2)

        Dim parent As String: parent = "TIER_" & Normalize(dens & "_" & layer
& "_" & tier)
        If Not Nodes.Exists(parent) Then
            Dim metaH As New Scripting.Dictionary
            metaH("Density") = dens: metaH("Layer") = layer: metaH("Tier") =
tier
            Ensure parent, "TIER_ROOT", dens & " | " & layer & " | " & tier,
"Tier", metaH
        End If

        Dim meta As New Scripting.Dictionary
        If Len(notes) > 0 Then meta("Notes") = notes
        Ensure parent & "_" & Normalize(plat), parent, plat, "Tier", meta
    Next r
End Sub

' --------- AP capabilities ----------
```

```
Private Sub BuildAPs()
    Ensure "AP_ROOT", "ROOT", "AP capabilities", "AP", Nothing
    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("AP_Capabilities")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row

    For r = 2 To last
        Dim ap$, caps$, radios$, rf$, cln$, fra$, mu$, ofdma$, mg$, ble$
        ap = CStr(ws.Cells(r, 1).Value2)
        caps = CStr(ws.Cells(r, 2).Value2)
        radios = CStr(ws.Cells(r, 3).Value2)
        rf = CStr(ws.Cells(r, 4).Value2)
        cln = CStr(ws.Cells(r, 5).Value2)
        fra = CStr(ws.Cells(r, 6).Value2)
        mu = CStr(ws.Cells(r, 7).Value2)
        ofdma = CStr(ws.Cells(r, 8).Value2)
        mg = CStr(ws.Cells(r, 9).Value2)
        ble = CStr(ws.Cells(r, 10).Value2)

        Dim meta As New Scripting.Dictionary
        meta("Capabilities") = caps
        meta("Radios") = radios
        meta("RF_ASIC") = rf
        meta("CleanAir") = cln
        meta("FRA") = fra
        meta("MU_MIMO") = mu
        meta("OFDMA") = ofdma
        meta("mGig") = mg
        meta("BLE/IoT") = ble

        Ensure "AP_" & Normalize(ap), "AP_ROOT", ap, "AP", meta
    Next r
End Sub

' --------- WLC catalog ----------
Private Sub BuildWLCs()
    Ensure "WLC_ROOT", "ROOT", "WLC profiles", "WLC", Nothing
    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("WLC_Profiles")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row

    For r = 2 To last
        Dim w$, dep$, topo$, ap&, cli&, thp$, ha$
        w = CStr(ws.Cells(r, 1).Value2)
        dep = CStr(ws.Cells(r, 2).Value2)
        topo = CStr(ws.Cells(r, 3).Value2)
        ap = CLng(Nz(ws.Cells(r, 4).Value2, 0))
        cli = CLng(Nz(ws.Cells(r, 5).Value2, 0))
        thp = CStr(ws.Cells(r, 6).Value2)
        ha = CStr(ws.Cells(r, 7).Value2)

        Dim meta As New Scripting.Dictionary
        meta("DeploymentModes") = dep
        meta("Topology") = topo
        meta("MaxAPs") = ap
        meta("MaxClients") = cli
        meta("ThroughputGbps") = thp
```

```
        meta("HAOptions") = ha

        Ensure "WLC_" & Normalize(w), "WLC_ROOT", w, "WLC", meta
    Next r
End Sub

' --------- Glossary ----------
Private Sub BuildGlossary()
    Ensure "GLOSS_ROOT", "ROOT", "Glossary", "Glossary", Nothing
    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("Glossary")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To last
        Dim t$, e$: t = CStr(ws.Cells(r, 1).Value2): e = CStr(ws.Cells(r,
2).Value2)
        Dim meta As New Scripting.Dictionary: meta("Expansion") = e
        Ensure "TERM_" & Normalize(t), "GLOSS_ROOT", t, "Glossary", meta
    Next r
End Sub

' --------- Site evaluator (algorigram) ----------
Private Sub EvaluateSites()
    Ensure "DESIGN_ROOT", "ROOT", "Design recommendations", "Recommendation",
Nothing
    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("Sites")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    If last < 2 Then Exit Sub

    For r = 2 To last
        Dim site$, dens$, tier$, apCount&, clients&, mode$, centr$, reqCSV$,
notes$
        site = CStr(ws.Cells(r, 1).Value2)
        dens = UCase$(CStr(ws.Cells(r, 2).Value2))         ' Medium | High
        tier = UCase$(CStr(ws.Cells(r, 3).Value2))         ' Enterprise |
Advanced | Mission
        apCount = CLng(Nz(ws.Cells(r, 4).Value2, 0))
        clients = CLng(Nz(ws.Cells(r, 5).Value2, 0))
        mode = CStr(ws.Cells(r, 6).Value2)
        centr = CStr(ws.Cells(r, 7).Value2)                ' Yes/No
        reqCSV = CStr(ws.Cells(r, 8).Value2)               ' feature list
        notes = CStr(ws.Cells(r, 9).Value2)

        Dim rec As Scripting.Dictionary: Set rec = RecommendStack(dens, tier)
        Dim apPick As Scripting.Dictionary: Set apPick = PickAP(reqCSV)
        Dim wlcPick As Scripting.Dictionary: Set wlcPick = PickWLC(apCount,
clients, centr)

        Dim meta As New Scripting.Dictionary
        meta("DensityLevel") = dens
        meta("HATier") = tier
        meta("AP_Count") = apCount
        meta("Clients") = clients
        meta("CentralizedPreferred") = centr
        meta("RequiredFeatures") = reqCSV
        meta("Notes") = notes
```

```
        MergeMeta meta, rec, "Platform_"
        MergeMeta meta, apPick, "AP_"
        MergeMeta meta, wlcPick, "WLC_"

        Ensure "SITE_" & Normalize(site), "DESIGN_ROOT", site,
"Recommendation", meta

        ' Gap findings for AP features
        If Not apPick Is Nothing Then
            Dim gaps As String: gaps = apPick("Gaps")
            If Len(gaps) > 0 Then
                AddFinding "SITE_" & Normalize(site) & "_AP_GAPS", "SITE_" &
Normalize(site), "AP feature gaps", DictKV("Missing", gaps))
            End If
        End If
    Next r
End Sub

' ---------- Recommenders ----------
Private Function RecommendStack(density$, tier$) As Scripting.Dictionary
    Dim layers: layers = Array(IIf(density = "MEDIUM", "Distribution",
"Core"), "Distribution", "Access", "WLC", "AP")
    Dim out As New Scripting.Dictionary

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("Platform_Tiers")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row

    Dim i&
    For i = LBound(layers) To UBound(layers)
        Dim pick$
        pick = FindPlatform(ws, densityProper(density$), layers(i),
tierProper(tier$))
        If Len(pick) > 0 Then out(layers(i)) = pick
    Next i
    Set RecommendStack = out
End Function

Private Function FindPlatform(ws As Worksheet, density$, layer$, tier$) As
String
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To last
        If ws.Cells(r, 1).Value2 = density And ws.Cells(r, 2).Value2 = layer
And ws.Cells(r, 3).Value2 = tier Then
            FindPlatform = CStr(ws.Cells(r, 4).Value2)
            Exit Function
        End If
    Next r
End Function

Private Function PickAP(reqCSV$) As Scripting.Dictionary
    Dim req() As String: req = SplitList(reqCSV$)
    Dim bestID$, bestScore&, gapsOut$

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("AP_Capabilities")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
```

```
    For r = 2 To last
        Dim ap$, caps$, rf$, cln$, fra$, mg$, radios$
        ap = CStr(ws.Cells(r, 1).Value2)
        caps = CStr(ws.Cells(r, 2).Value2)
        radios = CStr(ws.Cells(r, 3).Value2)
        rf = CStr(ws.Cells(r, 4).Value2)
        cln = CStr(ws.Cells(r, 5).Value2)
        fra = CStr(ws.Cells(r, 6).Value2)
        mg = CStr(ws.Cells(r, 9).Value2)

        Dim offer As New Scripting.Dictionary
        offer("RF_ASIC") = rf
        offer("CleanAir") = cln
        offer("FRA") = fra
        offer("mGig") = mg
        offer("Radios") = radios
        offer("CapabilitiesCSV") = caps

        Dim score&, gaps$: score = FeatureScore(req, offer, gaps)
        If score > bestScore Then
            bestScore = score: bestID = ap: gapsOut = gaps
        End If
    Next r

    If Len(bestID) = 0 Then Exit Function
    Dim d As New Scripting.Dictionary
    d("Series") = bestID
    d("Score") = CStr(bestScore)
    d("Gaps") = gapsOut
    Set PickAP = d
End Function

Private Function PickWLC(ap&, cli&, centralized$) As Scripting.Dictionary
    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("WLC_Profiles")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    Dim best$, headroom As Double: headroom = -1

    For r = 2 To last
        Dim w$, dep$, maxAP&, maxCli&, thp$
        w = CStr(ws.Cells(r, 1).Value2)
        dep = CStr(ws.Cells(r, 2).Value2)
        maxAP = CLng(Nz(ws.Cells(r, 4).Value2, 0))
        maxCli = CLng(Nz(ws.Cells(r, 5).Value2, 0))
        thp = CStr(ws.Cells(r, 6).Value2)

        If UCase$(centralized$) = "YES" Then
            If Len(thp) = 0 Then GoTo NextRow
        End If
        If (maxAP = 0 Or ap <= maxAP) And (maxCli = 0 Or cli <= maxCli) Then
            Dim h As Double: h = RatioHeadroom(ap, maxAP) +
RatioHeadroom(cli, maxCli)
            If h > headroom Then headroom = h: best = w
        End If
NextRow:
```

```
    Next r

    If Len(best) = 0 Then Exit Function
    Dim d As New Scripting.Dictionary
    d("Model") = best
    d("Headroom") = Format(headroom, "0.00")
    Set PickWLC = d
End Function

' ---------- Scoring & helpers ----------
Private Function FeatureScore(req() As String, offer As Scripting.Dictionary,
ByRef gaps As String) As Long
    Dim i&, s&, miss As String
    For i = LBound(req) To UBound(req)
        Dim k$: k = UCase$(Trim$(req(i)))
        If Len(k) = 0 Then GoTo NextReq
        Select Case k
            Case "RF_ASIC": s = s + IIf(Yes(offer("RF_ASIC")), 2, 0): If Not
Yes(offer("RF_ASIC")) Then miss = miss & "RF_ASIC;"
            Case "CLEANAIR": s = s + IIf(Yes(offer("CleanAir")), 2, 0): If
Not Yes(offer("CleanAir")) Then miss = miss & "CleanAir;"
            Case "FRA": s = s + IIf(Yes(offer("FRA")), 2, 0): If Not
Yes(offer("FRA")) Then miss = miss & "FRA;"
            Case "MGIG": s = s + IIf(Yes(offer("mGig")), 1, 0): If Not
Yes(offer("mGig")) Then miss = miss & "mGig;"
            Case "8X8": s = s + IIf(InStr(1, offer("Radios"), "8x8",
vbTextCompare) > 0, 1, 0): If InStr(1, offer("Radios"), "8x8", vbTextCompare)
= 0 Then miss = miss & "8x8;"
            Case Else
                ' generic capability check
                s = s + IIf(InStr(1, UCase$(offer("CapabilitiesCSV")), k,
vbTextCompare) > 0, 1, 0)
        End Select
NextReq:
    Next i
    gaps = miss
    FeatureScore = s
End Function

Private Function RatioHeadroom(val&, cap&) As Double
    If cap <= 0 Then RatioHeadroom = 0 Else RatioHeadroom = (cap - val) / cap
End Function

Private Function SplitList(s$) As String()
    Dim t$: t = Replace(Replace(s, ",", ";"), "|", ";")
    SplitList = Split(t, ";")
End Function

Private Function Yes(v As Variant) As Boolean
    Dim u$: u = UCase$(CStr(v))
    Yes = (u = "YES" Or u = "TRUE")
End Function

Private Function Nz(v As Variant, Optional def As Variant) As Variant
    If IsError(v) Or IsEmpty(v) Or v = "" Then Nz = def Else Nz = v
```

```
End Function

Private Sub MergeMeta(target As Scripting.Dictionary, src As
Scripting.Dictionary, Optional prefix$ = "")
    If src Is Nothing Then Exit Sub
    Dim k: For Each k In src.Keys: target(prefix & k) = src(k): Next k
End Sub

Private Sub AddFinding(id$, parent$, title$, meta As Scripting.Dictionary)
    Ensure id, parent, title, "Finding", meta
End Sub

Private Sub Ensure(id$, parent$, title$, kind$, meta As Scripting.Dictionary)
    If Nodes Is Nothing Then Set Nodes = New Scripting.Dictionary
    If Not Nodes.Exists(id) Then
        Dim n As cNode: Set n = New cNode
        n.ID = id: n.ParentID = parent: n.Title = title: n.Kind = kind
        If Not meta Is Nothing Then
            Dim k: For Each k In meta.Keys: n.Meta(k) = meta(k): Next k
        End If
        Nodes(id) = n
        If Len(parent) > 0 Then AddChild parent, id
    End If
End Sub

Private Sub AddChild(parent$, child$)
    If ParentMap Is Nothing Then Set ParentMap = New Scripting.Dictionary
    If Not ParentMap.Exists(parent) Then
        Dim c As New Collection: Set ParentMap(parent) = c
    End If
    ParentMap(parent).Add child
End Sub

Private Function Normalize(ByVal s As String) As String
    Dim t$: t = Trim$(s)
    t = Replace(t, " ", "_"): t = Replace(t, "/", "_"): t = Replace(t, "-",
"_")
    t = Replace(t, "(", "_"): t = Replace(t, ")", "_"): t = Replace(t, ".",
"_")
    Normalize = UCase$(t)
End Function

Private Function densityProper(u$) As String
    densityProper = IIf(UCase$(u) = "HIGH", "High", "Medium")
End Function

Private Function tierProper(u$) As String
    Select Case UCase$(u)
        Case "ADVANCED": tierProper = "Advanced"
        Case "MISSION": tierProper = "Mission Critical"
        Case Else: tierProper = "Enterprise"
    End Select
End Function
' UserForm: frmCampusDesign
Option Explicit
```

```
Private Sub UserForm_Initialize()
    On Error Resume Next
    lvMeta.ColumnHeaders.Clear
    lvMeta.ColumnHeaders.Add , , "Key", 220
    lvMeta.ColumnHeaders.Add , , "Value", 320
    On Error GoTo 0

    mCampusDesign.Build
    BuildTree
    lblSummary.Caption = CStr(mCampusDesign.Nodes.Count) & " nodes"
End Sub

Private Sub BuildTree()
    tvNav.Nodes.Clear
    Dim k
    For Each k In mCampusDesign.Nodes.Keys
        Dim n As cNode: Set n = mCampusDesign.Nodes(k)
        If Len(n.ParentID) = 0 Then
            tvNav.Nodes.Add , , n.ID, Prefix(n.Kind) & n.Title
            AddChildren n.ID
        End If
    Next k
    tvNav.ExpandAll
End Sub

Private Sub AddChildren(parent$)
    If Not mCampusDesign.ParentMap.Exists(parent) Then Exit Sub
    Dim ch As Collection: Set ch = mCampusDesign.ParentMap(parent)
    Dim i&
    For i = 1 To ch.Count
        Dim cid$: cid = ch(i)
        Dim n As cNode: Set n = mCampusDesign.Nodes(cid)
        tvNav.Nodes.Add n.ParentID, tvwChild, n.ID, Prefix(n.Kind) & n.Title
        AddChildren n.ID
    Next i
End Sub

Private Function Prefix(kind$) As String
    Select Case kind
        Case "Tier": Prefix = "[Tier] "
        Case "AP": Prefix = "[AP] "
        Case "WLC": Prefix = "[WLC] "
        Case "Recommendation": Prefix = "• "
        Case "Finding": Prefix = "• "
        Case "Glossary": Prefix = "[G] "
        Case Else: Prefix = ""
    End Select
End Function

Private Sub tvNav_NodeClick(ByVal Node As MSComctlLib.Node)
    ShowMeta Node.Key
End Sub

Private Sub ShowMeta(nodeID$)
```

```
    Dim n As cNode: Set n = mCampusDesign.Nodes(nodeID)
    lvMeta.ListItems.Clear
    AddRow "Title", n.Title
    AddRow "Kind", n.Kind
    Dim k
    For Each k In n.Meta.Keys
        AddRow CStr(k), CStr(n.Meta(k))
    Next k
End Sub

Private Sub AddRow(k$, v$)
    Dim it As ListItem
    Set it = lvMeta.ListItems.Add(, , k)
    it.SubItems(1) = v
End Sub
```

What this gives you

- Platform tier logigram for Medium and High designs (core/distribution/access/WLC/AP).
- AP capability map and feature-fit scoring against your RequiredFeatures list.
- WLC selection by AP/client capacity and centralized preference.
- Site recommendation node that consolidates stack/platform, AP pick with gaps, and WLC pick.
- Glossary node collection for instant term expansion in UI.

If you want, I'll add:

- An export macro that prints a one-page "Design Bill of Platforms" per site.
- A balancer that suggests AP series upgrades (e.g., 9115→9120→9130) when gaps exist.

Workbook schema

Create these sheets with the exact headers and sample rows.

DocMeta

- Columns: Key, Value

- Rows:
    - UniqueIdentifier, 240-61268959
    - Revision, 3
    - PageCount, 43
    - Disclosure, Controlled
    - ControlledCopyNote, When downloaded from WEB, user must verify authorized version
    - SectionList, 1. Introduction; 2. Supporting Clauses; 3. Environmental; 4. Network Application Detailed Design

EnvParams

- Columns: Param, Requirement, Notes
- Rows:
    - RatedEMI, High EMI error-free operation, IEC/industrial
    - TemperatureRange, Wide temperature operation, Per site spec
    - HumidityAltitude, High altitude/industrial rated, —
    - InstallationClass, Industrial installation, —
    - Frequency, Rated frequency, Grid standard

Architectures

- Columns: ArchitectureID, Topology, ControlRooms, Segregation, Redundancy, TimeSync, CoreDistAccess, Notes
- Rows:
    - ARCH_SINGLE, Single control room, 1, Segregated VLANs, Dual-homed rings, GPS+NTP/PTP, Star/Ring, Typical small/medium yard
    - ARCH_SEGREGATED, Segregated control rooms, 2, Physical/Logical segregation, Dual-homed rings+MSTP, GPS+NTP/PTP, Three-tier, Critical installations

PhysicalEnv

- Columns: Item, Requirement, Detail
- Rows:
    - EquipmentHousing, Cabinets/racks per standard, IP rating as required
    - CableEntryTermination, Gland plates, earthing, segregation, Copper/fiber mgmt
    - CopperCabling, Industrial-rated, shielded where needed
    - FiberCables, Single-mode/multi-mode per design, Splice trays, OTDR budget
    - FiberTermination, LC/SC per design, Patch panels
    - FiberPatchLeads, Match type, length control
    - Cooling, Rack/room cooling, Redundancy as needed
    - EnvMonitoring, Temperature/humidity/door sensors, SNMP/DI

Devices

- Columns: DeviceClass, Examples, NetworkRole, TimeSync, Criticality, Notes
- Rows:
    - ProtectionIED, Relay/Multifunction IEDs, Process/Station bus, PTP/NTP, High, IEC 61850
    - SubstationGateway, Protocol conversion, Northbound SCADA, NTP, High, DNP3/IEC
    - StationRTU, Telemetry I/O, SCADA, NTP, High, —
    - StationIED, Logic/control, Station bus, PTP/NTP, Medium, —
    - GPS_NTP, GPS receiver with NTP/PTP, Time master, GPS/PTP/NTP, High, Grandmaster/Server

- UFLS, Load shedding controller, Fast automation, PTP, High, Deterministic
- Meters, Energy meters, Data/logging, NTP, Medium, —
- EngLaptops, Engineering HMI, Maintenance, NTP, Low, Controlled access
- TestSets, Test equipment, Temporary, —, Low, Air gapped
- Teleprotection, Comms protection, Protection WAN, —, High, Deterministic/SDH/MPLS
- CBM, Condition monitoring, Analytics, NTP, Medium, —
- IPCameras, Video (future), OT/Physical sec, NTP, Low, Segregated VLAN
- HMI, Local HMI, Operations, NTP, High, —
- IPTelephony, Voice (future), Auxiliary, NTP, Low, Segregated VLAN
- Routers, Edge/WAN, Northbound, NTP, High, Dual WAN where needed
- DataServers, Historian/SCADA, Compute, NTP, High, Redundant
- EngServers, Tools/DTMs, Compute, NTP, Medium, Segregated access

ComplianceRules

- Columns: RuleID, Scope, Expression, Severity, Message
- Rows:
  - R_ENV_EMI, Env, RatedEMI=High EMI error-free operation, High, Must tolerate high EMI
  - R_ENV_TEMP, Env, TemperatureRange LIKE "*Wide*", Medium, Wide temp operation required
  - R_TIME_MASTER, Arch, TimeSync IN ("GPS+NTP/PTP","PTP"), High, GPS grandmaster and NTP/PTP required

- R_SEGREGATION, Arch, Segregation IN ("Physical/Logical segregation","Segregated VLANs"), High, Segregate process/station/aux networks
- R_FIBER_TERM, Phys, FiberTermination LIKE "*Patch*", Medium, Controlled fiber patching
- R_COOLING_RED, Phys, Cooling LIKE "*Redund*", Medium, Cooling redundancy recommended
- R_ENV_MON, Phys, EnvMonitoring LIKE "*SNMP*", Low, Environmental monitoring telemetry
- R_DEV_PROT_PTP, Dev, DeviceClass="ProtectionIED" AND TimeSync LIKE "*PTP*", High, Protection IEDs require PTP/61850 accuracy
- R_UFLS_DET, Dev, DeviceClass="UFLS" AND TimeSync LIKE "*PTP*", High, UFLS deterministic sync

Class model

```
' Class Module: cNode
Option Explicit
Public ID As String
Public ParentID As String
Public Title As String
Public Kind As String      ' Doc | Env | Arch | Phys | Dev | Rule | Finding
Public Meta As Scripting.Dictionary
Private Sub Class_Initialize()
    Set Meta = New Scripting.Dictionary
End Sub
' Module: mSubstation
Option Explicit

' References:
' - Microsoft Scripting Runtime
' - Microsoft Forms 2.0
' - Microsoft Windows Common Controls 6.0

Public Nodes As Scripting.Dictionary
Public ParentMap As Scripting.Dictionary

Public Sub Build()
    Set Nodes = New Scripting.Dictionary
    Set ParentMap = New Scripting.Dictionary

    BuildDoc
    BuildEnv
    BuildPhys
```

```
    BuildArch
    BuildDevices
    EvaluateCompliance
End Sub

Private Sub BuildDoc()
    Ensure "ROOT", "", "Substation Automation — Network Architecture and
Application Design (Transmission Substations)", "Doc", Nothing
    Ensure "DOC_META", "ROOT", "Document metadata", "Doc", Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("DocMeta")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To last
        Dim k$, v$: k = CStr(ws.Cells(r, 1).Value2): v = CStr(ws.Cells(r,
2).Value2)
        AddFinding "DOC_" & Normalize(k), "DOC_META", k, DictKV("Value", v))
    Next r
End Sub

Private Sub BuildEnv()
    Ensure "ENV_ROOT", "ROOT", "Environmental design parameters", "Env",
Nothing
    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("EnvParams")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To last
        Dim p$, req$, n$: p = CStr(ws.Cells(r, 1).Value2): req =
CStr(ws.Cells(r, 2).Value2): n = CStr(ws.Cells(r, 3).Value2)
        Dim meta As New Scripting.Dictionary
        meta("Requirement") = req: If Len(n) > 0 Then meta("Notes") = n
        Ensure "ENV_" & Normalize(p), "ENV_ROOT", p, "Env", meta
    Next r
End Sub

Private Sub BuildPhys()
    Ensure "PHYS_ROOT", "ROOT", "Physical environment", "Phys", Nothing
    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("PhysicalEnv")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To last
        Dim item$, req$, det$: item = CStr(ws.Cells(r, 1).Value2): req =
CStr(ws.Cells(r, 2).Value2): det = CStr(ws.Cells(r, 3).Value2)
        Dim meta As New Scripting.Dictionary
        meta("Requirement") = req: If Len(det) > 0 Then meta("Detail") = det
        Ensure "PHYS_" & Normalize(item), "PHYS_ROOT", item, "Phys", meta
    Next r
End Sub

Private Sub BuildArch()
    Ensure "ARCH_ROOT", "ROOT", "Network architectures", "Arch", Nothing
    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("Architectures")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To last
        Dim id$, top$, rooms&, seg$, red$, tsync$, cda$, notes$
        id = CStr(ws.Cells(r, 1).Value2)
        top = CStr(ws.Cells(r, 2).Value2)
        rooms = CLng(Nz(ws.Cells(r, 3).Value2, 0))
```

```
        seg = CStr(ws.Cells(r, 4).Value2)
        red = CStr(ws.Cells(r, 5).Value2)
        tsync = CStr(ws.Cells(r, 6).Value2)
        cda = CStr(ws.Cells(r, 7).Value2)
        notes = CStr(ws.Cells(r, 8).Value2)

        Dim meta As New Scripting.Dictionary
        meta("Topology") = top
        meta("ControlRooms") = rooms
        meta("Segregation") = seg
        meta("Redundancy") = red
        meta("TimeSync") = tsync
        meta("CoreDistAccess") = cda
        If Len(notes) > 0 Then meta("Notes") = notes

        Ensure "ARCH_" & Normalize(id), "ARCH_ROOT", id, "Arch", meta
    Next r
End Sub

Private Sub BuildDevices()
    Ensure "DEV_ROOT", "ROOT", "Connected devices", "Dev", Nothing
    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("Devices")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To last
        Dim cls$, ex$, role$, tsync$, crit$, notes$
        cls = CStr(ws.Cells(r, 1).Value2)
        ex = CStr(ws.Cells(r, 2).Value2)
        role = CStr(ws.Cells(r, 3).Value2)
        tsync = CStr(ws.Cells(r, 4).Value2)
        crit = CStr(ws.Cells(r, 5).Value2)
        notes = CStr(ws.Cells(r, 6).Value2)

        Dim meta As New Scripting.Dictionary
        meta("Examples") = ex
        meta("NetworkRole") = role
        meta("TimeSync") = tsync
        meta("Criticality") = crit
        If Len(notes) > 0 Then meta("Notes") = notes

        Ensure "DEV_" & Normalize(cls), "DEV_ROOT", cls, "Dev", meta
    Next r
End Sub

Private Sub EvaluateCompliance()
    Ensure "COMP_ROOT", "ROOT", "Compliance evaluation", "Finding", Nothing

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("ComplianceRules")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row

    Dim env As Scripting.Dictionary: Set env = Snapshot("EnvParams", "Param",
Array("Requirement"))
    Dim phys As Scripting.Dictionary: Set phys = Snapshot("PhysicalEnv",
"Item", Array("Requirement", "Detail"))
    Dim arch As Scripting.Dictionary: Set arch = Snapshot("Architectures",
"ArchitectureID", Array("Segregation", "TimeSync", "Topology"))
```

```
    Dim dev As Scripting.Dictionary: Set dev = Snapshot("Devices",
"DeviceClass", Array("TimeSync"))

    For r = 2 To last
        Dim rule$, scope$, expr$, sev$, msg$
        rule = CStr(ws.Cells(r, 1).Value2)
        scope = UCase$(CStr(ws.Cells(r, 2).Value2))
        expr = CStr(ws.Cells(r, 3).Value2)
        sev = CStr(ws.Cells(r, 4).Value2)
        msg = CStr(ws.Cells(r, 5).Value2)

        Dim ok As Boolean, detail$
        Select Case scope
            Case "ENV": ok = EvalEnv(expr, env, detail)
            Case "PHYS": ok = EvalPhys(expr, phys, detail)
            Case "ARCH": ok = EvalArch(expr, arch, detail)
            Case "DEV": ok = EvalDev(expr, dev, detail)
            Case Else: ok = False: detail = "Unknown scope"
        End Select

        Dim meta As New Scripting.Dictionary
        meta("Scope") = scope
        meta("Severity") = sev
        meta("Expression") = expr
        meta("Status") = IIf(ok, "PASS", "FAIL")
        meta("Message") = msg
        If Len(detail) > 0 Then meta("Detail") = detail

        Ensure "COMP_" & Normalize(rule), "COMP_ROOT", rule, "Finding", meta
    Next r
End Sub

' --------- Evaluators ----------
Private Function EvalEnv(expr$, env As Scripting.Dictionary, ByRef detail$)
As Boolean
    ' e.g., "RatedEMI=High EMI error-free operation"
    EvalEnv = KeyEquals(env, "Requirement", expr, detail)
End Function

Private Function EvalPhys(expr$, phys As Scripting.Dictionary, ByRef detail$)
As Boolean
    EvalPhys = KeyLike(phys, Array("Requirement", "Detail"), expr, detail)
End Function

Private Function EvalArch(expr$, arch As Scripting.Dictionary, ByRef detail$)
As Boolean
    ' e.g., "TimeSync IN (""GPS+NTP/PTP"",""PTP"")"
    If InStr(1, UCase$(expr), "IN", vbTextCompare) > 0 Then
        EvalArch = KeyIn(arch, "TimeSync", ParseIn(expr), detail)
    Else
        EvalArch = KeyLike(arch, Array("Segregation", "Topology",
"TimeSync"), expr, detail)
    End If
End Function
```

```
Private Function EvalDev(expr$, dev As Scripting.Dictionary, ByRef detail$)
As Boolean
    ' e.g., DeviceClass="ProtectionIED" AND TimeSync LIKE "*PTP*"
    Dim wantClass$, wantSync$
    wantClass = Between(expr, "DeviceClass=""", """")
    wantSync = After(expr, "TimeSync")
    If Len(wantClass) > 0 Then
        Dim row As Scripting.Dictionary
        If dev.Exists(wantClass) Then
            Set row = dev(wantClass)
            If InStr(1, UCase$(wantSync), "LIKE", vbTextCompare) > 0 Then
                Dim pat$: pat = Trim$(Replace(Split(wantSync, "LIKE")(1),
"*", ""))
                If InStr(1, UCase$(row("TimeSync")), UCase$(pat),
vbTextCompare) > 0 Then EvalDev = True Else detail = row("TimeSync")
            Else
                EvalDev = (UCase$(row("TimeSync")) = UCase$(wantSync))
            End If
        Else
            detail = "DeviceClass not found"
        End If
    End If
End Function

' --------- Snapshots and helpers ----------
Private Function Snapshot(sheetName$, keyCol$, valCols As Variant) As
Scripting.Dictionary
    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets(sheetName)
    Dim d As New Scripting.Dictionary, r&, last&: last =
ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    Dim keyIndex&, i&
    keyIndex = ColumnIndex(ws, keyCol$)
    For r = 2 To last
        Dim k$: k = CStr(ws.Cells(r, keyIndex).Value2)
        If Len(k) = 0 Then GoTo NextR
        Dim row As New Scripting.Dictionary
        For i = LBound(valCols) To UBound(valCols)
            Dim c$: c = CStr(valCols(i))
            row(c) = CStr(ws.Cells(r, ColumnIndex(ws, c)).Value2)
        Next i
        d(k) = row
NextR:
    Next r
    Set Snapshot = d
End Function

Private Function ColumnIndex(ws As Worksheet, header$) As Long
    Dim c&: For c = 1 To ws.UsedRange.Columns.Count
        If UCase$(CStr(ws.Cells(1, c).Value2)) = UCase$(header$) Then
ColumnIndex = c: Exit Function
    Next c
End Function

Private Function KeyEquals(d As Scripting.Dictionary, field$, expr$, ByRef
detail$) As Boolean
```

```
    ' pattern "Key=Value"
    Dim k$: k = Split(expr$, "=")(0)
    Dim v$: v = Mid$(expr$, Len(k) + 2)
    If d.Exists(k) Then
        Dim row As Scripting.Dictionary: Set row = d(k)
        KeyEquals = (row(field$) = v)
        If Not KeyEquals Then detail = row(field$)
    Else
        detail = "Key not found: " & k
    End If
End Function

Private Function KeyLike(d As Scripting.Dictionary, fields As Variant, expr$,
ByRef detail$) As Boolean
    ' pattern "Field LIKE ""*text*"""
    Dim tgtField$, pat$
    If InStr(1, UCase$(expr$), "LIKE", vbTextCompare) = 0 Then KeyLike =
False: detail = "Unsupported expr": Exit Function
    tgtField = Trim$(Split(expr$, "LIKE")(0))
    pat = Between(expr$, """", """", True)
    Dim k: For Each k In d.Keys
        Dim row As Scripting.Dictionary: Set row = d(k)
        Dim i&: For i = LBound(fields) To UBound(fields)
            If UCase$(fields(i)) = UCase$(tgtField) Then
                If LikeText(row(CStr(fields(i))), pat) Then KeyLike = True:
Exit Function
            End If
        Next i
    Next k
    detail = "No match for " & tgtField & " LIKE " & pat
End Function

Private Function KeyIn(d As Scripting.Dictionary, field$, values As
Collection, ByRef detail$) As Boolean
    Dim k: For Each k In d.Keys
        Dim row As Scripting.Dictionary: Set row = d(k)
        If values.Contains(UCase$(row(field$))) Then KeyIn = True: Exit
Function
    Next k
    detail = "No value in set"
End Function

Private Function ParseIn(expr$) As Collection
    Dim c As New Collection, inner$: inner = Between(expr$, "(", ")", True)
    Dim parts() As String: parts = Split(inner, ",")
    Dim i&: For i = LBound(parts) To UBound(parts)
        c.Add UCase$(Trim$(Replace(Replace(parts(i), """", ""), "'", "")))
    Next i
    Set ParseIn = c
End Function

Private Function Between(s$, a$, b$, Optional lastPair As Boolean = False) As
String
    Dim p&, q&
    p = InStr(1, s, a)
```

```
    If p = 0 Then Exit Function
    If lastPair Then
        q = InStrRev(s, b)
    Else
        q = InStr(p + Len(a), s, b)
    End If
    If q > p Then Between = Mid$(s, p + Len(a), q - (p + Len(a)))
End Function

Private Function After(s$, token$) As String
    Dim p&: p = InStr(1, UCase$(s), UCase$(token$))
    If p = 0 Then Exit Function
    After = Mid$(s, p + Len(token$))
End Function

Private Function LikeText(val$, pat$) As Boolean
    ' emulate LIKE with wildcard *
    Dim uVal$: uVal = UCase$(val$)
    Dim uPat$: uPat = UCase$(Replace(pat$, "*", ""))
    LikeText = (InStr(1, uVal, uPat, vbTextCompare) > 0)
End Function

' --------- Node helpers ----------
Private Sub AddFinding(id$, parent$, title$, meta As Scripting.Dictionary)
    Ensure id, parent, title, "Finding", meta
End Sub

Public Sub Ensure(id$, parent$, title$, kind$, meta As Scripting.Dictionary)
    If Nodes Is Nothing Then Set Nodes = New Scripting.Dictionary
    If Not Nodes.Exists(id) Then
        Dim n As cNode: Set n = New cNode
        n.ID = id: n.ParentID = parent: n.Title = title: n.Kind = kind
        If Not meta Is Nothing Then
            Dim k: For Each k In meta.Keys: n.Meta(k) = meta(k): Next k
        End If
        Nodes(id) = n
        If Len(parent$) > 0 Then AddChild parent$, id$
    End If
End Sub

Private Sub AddChild(parent$, child$)
    If ParentMap Is Nothing Then Set ParentMap = New Scripting.Dictionary
    If Not ParentMap.Exists(parent$) Then
        Dim c As New Collection: Set ParentMap(parent$) = c
    End If
    ParentMap(parent$).Add child$
End Sub

Private Function Normalize(ByVal s As String) As String
    Dim t$: t = Trim$(s)
    t = Replace(t, " ", "_"): t = Replace(t, "/", "_"): t = Replace(t, "-",
"_")
    t = Replace(t, "(", "_"): t = Replace(t, ")", "_"): t = Replace(t, ".",
"_")
    Normalize = UCase$(t)
```

```
End Function

Private Function Nz(v As Variant, Optional def As Variant) As Variant
    If IsError(v) Or IsEmpty(v) Or v = "" Then Nz = def Else Nz = v
End Function
' UserForm: frmSubstation
Option Explicit

Private Sub UserForm_Initialize()
    On Error Resume Next
    lvMeta.ColumnHeaders.Clear
    lvMeta.ColumnHeaders.Add , , "Key", 220
    lvMeta.ColumnHeaders.Add , , "Value", 380
    On Error GoTo 0

    mSubstation.Build
    BuildTree
    lblSummary.Caption = CStr(mSubstation.Nodes.Count) & " nodes loaded"
End Sub

Private Sub BuildTree()
    tvNav.Nodes.Clear
    Dim k
    For Each k In mSubstation.Nodes.Keys
        Dim n As cNode: Set n = mSubstation.Nodes(k)
        If Len(n.ParentID) = 0 Then
            tvNav.Nodes.Add , , n.ID, Prefix(n.Kind) & n.Title
            AddChildren n.ID
        End If
    Next k
    tvNav.ExpandAll
End Sub

Private Sub AddChildren(parent$)
    If Not mSubstation.ParentMap.Exists(parent$) Then Exit Sub
    Dim ch As Collection: Set ch = mSubstation.ParentMap(parent$)
    Dim i&
    For i = 1 To ch.Count
        Dim cid$: cid = ch(i)
        Dim n As cNode: Set n = mSubstation.Nodes(cid)
        tvNav.Nodes.Add n.ParentID, tvwChild, n.ID, Prefix(n.Kind) & n.Title
        AddChildren n.ID
    Next i
End Sub

Private Function Prefix(kind$) As String
    Select Case kind$
        Case "Doc": Prefix = "[DOC] "
        Case "Env": Prefix = "[ENV] "
        Case "Phys": Prefix = "[PHYS] "
        Case "Arch": Prefix = "[ARCH] "
        Case "Dev": Prefix = "[DEV] "
        Case "Finding": Prefix = "• "
        Case Else: Prefix = ""
    End Select
```

```
End Function

Private Sub tvNav_NodeClick(ByVal Node As MSComctlLib.Node)
    ShowMeta Node.Key
End Sub

Private Sub ShowMeta(nodeID$)
    Dim n As cNode: Set n = mSubstation.Nodes(nodeID$)
    lvMeta.ListItems.Clear
    AddRow "Title", n.Title
    AddRow "Kind", n.Kind

    Dim k
    For Each k In n.Meta.Keys
        AddRow CStr(k), CStr(n.Meta(k))
    Next k
End Sub

Private Sub AddRow(k$, v$)
    Dim it As ListItem
    Set it = lvMeta.ListItems.Add(, , k$)
    it.SubItems(1) = v$
End Sub
```

Built-in checks and findings

- **Environmental:** High EMI tolerance, wide temperature, industrial installation.
- **Architecture:** Enforces segregation (physical/logical), dual-homing/rings, time synchronization (GPS NTP/PTP).
- **Physical:** Fiber termination and patching practices, cooling redundancy, environmental monitoring telemetry.
- **Devices:** Protection IEDs and UFLS require deterministic time (PTP/IEC 61850 class); gateways/RTUs/servers on NTP.
- **Document control:** Surfaces controlled disclosure and “verify authorized version” note.

Fast start

1. Insert class cNode, module mSubstation, and UserForm frmSubstation.
2. Set references: Microsoft Scripting Runtime, Forms 2.0, Windows Common Controls 6.0.

3. Create and seed sheets: DocMeta, EnvParams, Architectures, PhysicalEnv, Devices, ComplianceRules.

4. Run: Immediate Window → `VBA.UserForms.Add("frmSubstation").Show`.

Want me to add an export that generates a one-page compliance statement (PASS/FAIL by scope with remediation notes) for tender or audit packs?

Reusable VBA logigram and algorigram scaffold

Drop-in Excel VBA framework to build:

- A logigram (hierarchical map of anything: systems, devices, standards).
- An algorigram (rule checks over that map, with PASS/FAIL findings).

It's domain-agnostic. Populate two simple sheets and you've got a navigable TreeView with rule results.

1) Workbook schema

Create these sheets with exact headers.

Sheet: Nodes

- Columns:
  - ID
  - ParentID
  - Title
  - Kind
- Example rows:
  - SYS | | Substation Automation | Domain
  - ENV | SYS | Environmental design | Section
  - DEV | SYS | Devices | Section
  - IED_PROT | DEV | Protection IEDs | Device

- NTP_GPS | DEV | GPS + NTP/PTP | TimeSync

Sheet: NodeMeta

- Columns:
  - NodeID
  - Key
  - Value
- Example rows:
  - ENV | RatedEMI | High
  - ENV | TemperatureRange | Wide
  - IED_PROT | TimeSync | PTP
  - NTP_GPS | Role | Grandmaster

Sheet: Rules

- Columns:
  - RuleID
  - Target (NodeID or Kind or “ALL”)
  - Expression (simple DSL; see below)
  - Severity (High/Medium/Low)
  - Message
- Example rows:
  - R1 | Kind=Device | TimeSync LIKE "*PTP*" | High | Protection devices require PTP
  - R2 | NodeID=ENV | RatedEMI="High" AND TemperatureRange LIKE "*Wide*" | Medium | Environmental envelope not met if false
  - R3 | ALL | Role IN ("Grandmaster","Server") OR TimeSync LIKE "*NTP*" | Low | Time service should be present

Expression operators supported (case-insensitive):

- Comparators: =, <>, >, >=, <, <= (numeric only)
- LIKE with “*” wildcard (text)
- IN ("A","B","C") set membership (text)
- AND / OR (left-to-right; no parentheses)
- Left operand keys must exist in NodeMeta (by NodeID). Nonexistent keys evaluate as empty strings.

2) Class: cNode

vba

```
' Class Module: cNode
Option Explicit
Public ID As String
Public ParentID As String
Public Title As String
Public Kind As String
Public Meta As Scripting.Dictionary

Private Sub Class_Initialize()
    Set Meta = New Scripting.Dictionary
End Sub
```

3) Engine: mLogiAlgo

```
' Module: mLogiAlgo
Option Explicit

' References:
' - Microsoft Scripting Runtime
' - Microsoft Forms 2.0
' - Microsoft Windows Common Controls 6.0 (SP6)

Public Nodes As Scripting.Dictionary          ' ID -> cNode
Public ParentMap As Scripting.Dictionary       ' ParentID ->
Collection(childIDs)
Public Rules As Collection                     ' of RuleRec

Private Type RuleRec
    RuleID As String
    TargetType As String ' NODEID | KIND | ALL
    TargetValue As String
    Expression As String
    Severity As String
```

```
    Message As String
End Type

Public Sub Build()
    Set Nodes = New Scripting.Dictionary
    Set ParentMap = New Scripting.Dictionary
    Set Rules = New Collection

    LoadNodes
    LoadMeta
    LoadRules
End Sub

Private Sub LoadNodes()
    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("Nodes")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To last
        Dim id$, pid$, ttl$, kind$
        id = CStr(ws.Cells(r, 1).Value2)
        If Len(id) = 0 Then GoTo NextR
        pid = CStr(ws.Cells(r, 2).Value2)
        ttl = CStr(ws.Cells(r, 3).Value2)
        kind = CStr(ws.Cells(r, 4).Value2)

        Dim n As New cNode
        n.ID = id: n.ParentID = pid: n.Title = ttl: n.Kind = kind
        Nodes(id) = n
        If Len(pid) > 0 Then AddChild pid, id
NextR:
    Next r
End Sub

Private Sub LoadMeta()
    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("NodeMeta")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To last
        Dim nid$, k$, v$
        nid = CStr(ws.Cells(r, 1).Value2)
        If Len(nid) = 0 Then GoTo NextR
        k = CStr(ws.Cells(r, 2).Value2)
        v = CStr(ws.Cells(r, 3).Value2)
        If Nodes.Exists(nid) And Len(k) > 0 Then Nodes(nid).Meta(k) = v
NextR:
    Next r
End Sub

Private Sub LoadRules()
    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("Rules")
    Dim r&, last&: last = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To last
        Dim rr As RuleRec, tgt$
        rr.RuleID = CStr(ws.Cells(r, 1).Value2)
        tgt = CStr(ws.Cells(r, 2).Value2)
        rr.Expression = CStr(ws.Cells(r, 3).Value2)
        rr.Severity = CStr(ws.Cells(r, 4).Value2)
```

```
        rr.Message = CStr(ws.Cells(r, 5).Value2)
        ParseTarget tgt, rr.TargetType, rr.TargetValue
        If Len(rr.RuleID) > 0 Then Rules.Add rr
    Next r
End Sub

Private Sub ParseTarget(ByVal raw$, ByRef tType$, ByRef tVal$)
    Dim u$: u = UCase$(Trim$(raw$))
    If Left$(u, 7) = "NODEID=" Then tType = "NODEID": tVal = Mid$(raw$, 8):
Exit Sub
    If Left$(u, 5) = "KIND=" Then tType = "KIND": tVal = Mid$(raw$, 6): Exit
Sub
    If u = "ALL" Or u = "" Then tType = "ALL": tVal = "": Exit Sub
    ' default: treat as KIND
    tType = "KIND": tVal = raw$
End Sub

Private Sub AddChild(parentID$, childID$)
    If Not ParentMap.Exists(parentID$) Then
        Dim c As New Collection: Set ParentMap(parentID$) = c
    End If
    ParentMap(parentID$).Add childID$
End Sub

' -------- Evaluation --------

Public Function EvaluateAll() As Scripting.Dictionary
    ' Returns: Dict key = NodeID, value = Collection of findings (each dict
with RuleID, Severity, Status, Message)
    Dim out As New Scripting.Dictionary
    Dim k: For Each k In Nodes.Keys
        Dim findings As Collection
        Set findings = EvaluateNode(Nodes(CStr(k)))
        out(CStr(k)) = findings
    Next k
    Set EvaluateAll = out
End Function

Public Function EvaluateNode(n As cNode) As Collection
    Dim col As New Collection
    Dim i&, rr As RuleRec
    For i = 1 To Rules.Count
        rr = Rules(i)
        If RuleTargetsNode(rr, n) Then
            Dim pass As Boolean, detail$
            pass = EvalExpr(rr.Expression, n.Meta, detail)
            Dim f As Scripting.Dictionary: Set f = New Scripting.Dictionary
            f("RuleID") = rr.RuleID
            f("Severity") = rr.Severity
            f("Status") = IIf(pass, "PASS", "FAIL")
            f("Message") = rr.Message
            If Len(detail) > 0 Then f("Detail") = detail
            col.Add f
        End If
    Next i
```

```
    Set EvaluateNode = col
End Function

Private Function RuleTargetsNode(rr As RuleRec, n As cNode) As Boolean
    Select Case rr.TargetType
        Case "ALL": RuleTargetsNode = True
        Case "NODEID": RuleTargetsNode = (StrComp(n.ID, rr.TargetValue,
vbTextCompare) = 0)
        Case "KIND": RuleTargetsNode = (StrComp(n.Kind, rr.TargetValue,
vbTextCompare) = 0)
        Case Else: RuleTargetsNode = False
    End Select
End Function

' -------- Expression evaluator (simple DSL) --------
Private Function EvalExpr(expr$, meta As Scripting.Dictionary, ByRef detail$)
As Boolean
    ' Supports AND/OR (left-to-right), =, <>, >, >=, <, <=, LIKE "*", IN
("a","b")
    Dim tokens() As String: tokens = Tokenize(expr$)
    If UBound(tokens) < 0 Then EvalExpr = True: Exit Function
    Dim i&, cur As Variant, op$, nextVal As Variant, res As Variant
    res = True: op = "AND"

    i = 0
    Do While i <= UBound(tokens)
        Dim lhs$, oper$, rhs$
        lhs = tokens(i): i = i + 1
        If i > UBound(tokens) Then Exit Do
        oper = UCase$(tokens(i)): i = i + 1

        ' RHS may be a value, a quoted string, an IN (...) or LIKE pattern
segment
        If oper = "IN" Then
            rhs = ReadParenList(tokens, i) ' returns CSV of uppercased values
        Else
            If i <= UBound(tokens) Then
                rhs = tokens(i): i = i + 1
            End If
        End If

        Dim test As Boolean
        test = EvalOne(lhs, oper, rhs, meta, detail)

        If op = "AND" Then
            res = (res And test)
        ElseIf op = "OR" Then
            res = (res Or test)
        End If

        ' Next logical operator if present
        If i <= UBound(tokens) Then
            Dim maybeOp$: maybeOp = UCase$(tokens(i))
            If maybeOp = "AND" Or maybeOp = "OR" Then
                op = maybeOp: i = i + 1
```

```
            End If
        End If
    Loop

    EvalExpr = CBool(res)
End Function

Private Function EvalOne(lhs$, oper$, rhs$, meta As Scripting.Dictionary,
ByRef detail$) As Boolean
    Dim lval$, uoper$
    lval = GetMeta(meta, lhs$)
    uoper = UCase$(oper$)

    Select Case uoper
        Case "=": EvalOne = (Norm(lval) = Norm(Unquote(rhs$)))
        Case "<>": EvalOne = (Norm(lval) <> Norm(Unquote(rhs$)))
        Case "LIKE": EvalOne = LikeMatch(lval, Unquote(rhs$))
        Case "IN"
            EvalOne = InCSV(UCase$(Norm(lval)), rhs$)
        Case ">", ">=", "<", "<="
            If IsNumeric(lval) And IsNumeric(rhs$) Then
                Dim a#, b#: a = CDbl(lval): b = CDbl(rhs$)
                Select Case uoper
                    Case ">": EvalOne = (a > b)
                    Case ">=": EvalOne = (a >= b)
                    Case "<": EvalOne = (a < b)
                    Case "<=": EvalOne = (a <= b)
                End Select
            Else
                detail = "Non-numeric compare: " & lhs$
                EvalOne = False
            End If
        Case Else
            detail = "Unsupported operator: " & oper$
            EvalOne = False
    End Select
End Function

Private Function GetMeta(meta As Scripting.Dictionary, key$) As String
    Dim k$: k = Trim$(key$)
    If meta.Exists(k) Then
        GetMeta = CStr(meta(k))
    Else
        GetMeta = ""
    End If
End Function

Private Function Norm(s$) As String
    Norm = Trim$(CStr(s$))
End Function

Private Function Unquote(s$) As String
    If Len(s$) >= 2 Then
        If (Left$(s$, 1) = """" And Right$(s$, 1) = """") Or (Left$(s$, 1) =
"'" And Right$(s$, 1) = "'") Then
```

```
            Unquote = Mid$(s$, 2, Len(s$) - 2): Exit Function
        End If
    End If
    Unquote = s$
End Function

Private Function LikeMatch(val$, pat$) As Boolean
    LikeMatch = (UCase$(val$) Like UCase$(pat$))
End Function

Private Function InCSV(uVal$, csvUpperList$) As Boolean
    ' csvUpperList is "A;B;C" uppercased by ReadParenList
    Dim arr() As String: arr = Split(csvUpperList$, ";")
    Dim i&: For i = LBound(arr) To UBound(arr)
        If uVal$ = Trim$(arr(i)) Then InCSV = True: Exit Function
    Next i
End Function

Private Function Tokenize(expr$) As String()
    Dim s$: s = Trim$(expr$)
    Dim out() As String: ReDim out(0 To -1)
    Dim i&, cur$, ch$
    i = 1
    Do While i <= Len(s)
        ch = Mid$(s, i, 1)
        Select Case ch
            Case " "
                If Len(cur) > 0 Then Push out, cur: cur = ""
            Case """", "'"
                Dim q$: q = ch: cur = cur & ch: i = i + 1
                Do While i <= Len(s) And Mid$(s, i, 1) <> q
                    cur = cur & Mid$(s, i, 1): i = i + 1
                Loop
                If i <= Len(s) Then cur = cur & q
                Push out, cur: cur = ""
            Case "("
                Push out, cur: cur = "("
                i = i + 1
                Dim depth&: depth = 1
                Do While i <= Len(s) And depth > 0
                    ch = Mid$(s, i, 1)
                    cur = cur & ch
                    If ch = "(" Then depth = depth + 1
                    If ch = ")" Then depth = depth - 1
                    i = i + 1
                Loop
                Push out, cur: cur = ""
            Case ","
                If Len(cur) > 0 Then Push out, cur: cur = ""
                Push out, ","
            Default
                cur = cur & ch
        End Select
        i = i + 1
    Loop
```

```
    If Len(cur) > 0 Then Push out, cur

    Tokenize = out
End Function

Private Function ReadParenList(tokens() As String, ByRef i&) As String
    ' Expects current tokens(i) to be a list starting with "(" and ending
with ")"
    Dim raw$: raw = tokens(i)
    ' Strip parentheses and quotes; return uppercased semicolon list
    raw = Replace(raw, "(", "")
    raw = Replace(raw, ")", "")
    raw = Replace(raw, """", "")
    raw = Replace(raw, "'", "")
    raw = Trim$(raw)
    raw = Replace(raw, ",", ";")
    i = i ' position already consumed in caller
    ReadParenList = UCase$(raw)
End Function

Private Sub Push(ByRef arr() As String, ByVal s$)
    Dim n&: n = UBound(arr) + 1
    ReDim Preserve arr(0 To n)
    arr(n) = Trim$(s$)
End Sub
```

4) UserForm: frmLogiAlg

```
' UserForm: frmLogiAlgo
Option Explicit

Private Sub UserForm_Initialize()
    On Error Resume Next
    lvMeta.ColumnHeaders.Clear
    lvMeta.ColumnHeaders.Add , , "Key", 200
    lvMeta.ColumnHeaders.Add , , "Value", 320
    On Error GoTo 0

    mLogiAlgo.Build
    BuildTree
    lblSummary.Caption = CStr(mLogiAlgo.Nodes.Count) & " nodes loaded"
End Sub

Private Sub BuildTree()
    tvNav.Nodes.Clear
    ' Rootless nodes at top level
    Dim k
    For Each k In mLogiAlgo.Nodes.Keys
        Dim n As cNode: Set n = mLogiAlgo.Nodes(k)
        If Len(n.ParentID) = 0 Then
            tvNav.Nodes.Add , , n.ID, Prefix(n.Kind) & n.Title
            AddChildren n.ID
        End If
    Next k
```

```
    tvNav.ExpandAll
End Sub

Private Sub AddChildren(parentID$)
    If Not mLogiAlgo.ParentMap.Exists(parentID$) Then Exit Sub
    Dim ch As Collection: Set ch = mLogiAlgo.ParentMap(parentID$)
    Dim i&
    For i = 1 To ch.Count
        Dim cid$: cid = ch(i)
        Dim n As cNode: Set n = mLogiAlgo.Nodes(cid)
        tvNav.Nodes.Add n.ParentID, tvwChild, n.ID, Prefix(n.Kind) & n.Title
        AddChildren n.ID
    Next i
End Sub

Private Function Prefix(kind$) As String
    If Len(kind$) = 0 Then Prefix = "" Else Prefix = "[" & kind$ & "] "
End Function

Private Sub tvNav_NodeClick(ByVal Node As MSComctlLib.Node)
    ShowNode Node.Key
End Sub

Private Sub ShowNode(nodeID$)
    Dim n As cNode: Set n = mLogiAlgo.Nodes(nodeID$)
    lvMeta.ListItems.Clear
    AddRow "Title", n.Title
    AddRow "Kind", n.Kind
    Dim k
    For Each k In n.Meta.Keys
        AddRow CStr(k), CStr(n.Meta(k))
    Next k

    ' Findings
    Dim results As Collection: Set results = mLogiAlgo.EvaluateNode(n)
    Dim j&: For j = 1 To results.Count
        Dim f As Scripting.Dictionary: Set f = results(j)
        AddRow "— Finding " & CStr(j), f("RuleID") & " [" & f("Severity") &
"] " & f("Status")
        AddRow "  Message", f("Message")
        If f.Exists("Detail") Then AddRow "  Detail", f("Detail")
    Next j
End Sub

Private Sub AddRow(k$, v$)
    Dim it As ListItem
    Set it = lvMeta.ListItems.Add(, , k$)
    it.SubItems(1) = v$
End Sub
```

5) How to run

1. Insert:

    - Class module cNode.
    - Standard module mLogiAlgo.
    - UserForm frmLogiAlgo (TreeView/ListView/Label as above).
2. Set references:
    - Microsoft Scripting Runtime
    - Microsoft Forms 2.0 Object Library
    - Microsoft Windows Common Controls 6.0 (SP6)
3. Create sheets: Nodes, NodeMeta, Rules. Seed with your data.
4. Run:
    - Immediate window: `VBA.UserForms.Add("frmLogiAlgo").Show`

6) Extend in seconds

- Add more meta to any Node in NodeMeta (no code changes).
- Add rules for any Kind or specific NodeID.
- Copy this workbook and reuse for different domains (UPS, LTMR, campus LAN, substation standards).
- Perfect—you're ready to implement a full VBA logigram and algorigram engine based on your thermodynamics and control systems schema. Here's a complete Visual Basic for Applications (VBA) code package that reads your workbook structure, builds a hierarchical map of concepts, and evaluates rule-based logic across formulas and system blocks.

• □ Step 1: Class Module – cConcept

- Insert a class module named `cConcept`:
- vba
- ```
  ' Class Module: cConcept
  Option Explicit
  
  Public ID As String
  Public ParentID As String
  Public Title As String
  ```
  ```

```
Public Kind As String
Public Meta As Scripting.Dictionary

Private Sub Class_Initialize()
    Set Meta = New Scripting.Dictionary
End Sub
' Module: mConceptEngine
Option Explicit

Public Concepts As Scripting.Dictionary
Public ParentMap As Scripting.Dictionary
Public Rules As Collection

Private Type RuleRec
    RuleID As String
    TargetType As String ' ConceptID | Kind | ALL
    TargetValue As String
    Expression As String
    Severity As String
    Message As String
End Type

Public Sub Build()
    Set Concepts = New Scripting.Dictionary
    Set ParentMap = New Scripting.Dictionary
    Set Rules = New Collection

    LoadConcepts
    LoadMeta
    LoadRules
End Sub

Private Sub LoadConcepts()
    Dim ws As Worksheet: Set ws = ThisWorkbook.Sheets("Concepts")
    Dim r&, lastRow&: lastRow = ws.Cells(ws.Rows.Count,
1).End(xlUp).Row

    For r = 2 To lastRow
        Dim id$, pid$, ttl$, kind$
        id = ws.Cells(r, 1).Value2
        pid = ws.Cells(r, 2).Value2
        ttl = ws.Cells(r, 3).Value2
        kind = ws.Cells(r, 4).Value2

        Dim c As New cConcept
        c.ID = id: c.ParentID = pid: c.Title = ttl: c.Kind = kind
        Concepts(id) = c
        If Len(pid) > 0 Then AddChild pid, id
    Next r
End Sub

Private Sub LoadMeta()
    Dim ws As Worksheet: Set ws = ThisWorkbook.Sheets("ConceptMeta")
    Dim r&, lastRow&: lastRow = ws.Cells(ws.Rows.Count,
1).End(xlUp).Row
```

```
    For r = 2 To lastRow
        Dim cid$, k$, v$
        cid = ws.Cells(r, 1).Value2
        k = ws.Cells(r, 2).Value2
        v = ws.Cells(r, 3).Value2
        If Concepts.Exists(cid) Then Concepts(cid).Meta(k) = v
    Next r
End Sub

Private Sub LoadRules()
    Dim ws As Worksheet: Set ws = ThisWorkbook.Sheets("Rules")
    Dim r&, lastRow&: lastRow = ws.Cells(ws.Rows.Count,
1).End(xlUp).Row

    For r = 2 To lastRow
        Dim rr As RuleRec
        rr.RuleID = ws.Cells(r, 1).Value2
        Dim tgt$: tgt = ws.Cells(r, 2).Value2
        rr.Expression = ws.Cells(r, 3).Value2
        rr.Severity = ws.Cells(r, 4).Value2
        rr.Message = ws.Cells(r, 5).Value2
        ParseTarget tgt, rr.TargetType, rr.TargetValue
        Rules.Add rr
    Next r
End Sub

Private Sub ParseTarget(ByVal raw$, ByRef tType$, ByRef tVal$)
    If InStr(raw$, "=") > 0 Then
        tType = Split(raw$, "=")(0)
        tVal = Split(raw$, "=")(1)
    Else
        tType = "ALL": tVal = ""
    End If
End Sub

Private Sub AddChild(parent$, child$)
    If Not ParentMap.Exists(parent) Then
        Dim c As New Collection: Set ParentMap(parent) = c
    End If
    ParentMap(parent).Add child
End Sub

Public Function EvaluateConcept(c As cConcept) As Collection
    Dim col As New Collection
    Dim i&, rr As RuleRec

    For i = 1 To Rules.Count
        rr = Rules(i)
        If RuleApplies(rr, c) Then
            Dim result As Scripting.Dictionary: Set result = New
Scripting.Dictionary
            result("RuleID") = rr.RuleID
            result("Severity") = rr.Severity
            result("Message") = rr.Message
```

```
            result("Status") = IIf(EvalExpr(rr.Expression, c.Meta), "PASS", "FAIL")
            col.Add result
        End If
    Next i
    Set EvaluateConcept = col
End Function

Private Function RuleApplies(rr As RuleRec, c As cConcept) As Boolean
    Select Case UCase(rr.TargetType)
        Case "ALL": RuleApplies = True
        Case "CONCEPTID": RuleApplies = (UCase(rr.TargetValue) = UCase(c.ID))
        Case "KIND": RuleApplies = (UCase(rr.TargetValue) = UCase(c.Kind))
        Case Else: RuleApplies = False
    End Select
End Function

Private Function EvalExpr(expr$, meta As Scripting.Dictionary) As Boolean
    Dim parts(): parts = Split(expr$, " ")
    If UBound(parts) < 2 Then EvalExpr = False: Exit Function

    Dim key$, op$, val$
    key = parts(0): op = UCase(parts(1)): val = Replace(parts(2), """", "")

    If Not meta.Exists(key) Then EvalExpr = False: Exit Function
    Dim actual$: actual = meta(key)

    Select Case op
        Case "=": EvalExpr = (actual = val)
        Case "LIKE": EvalExpr = (InStr(1, actual, val, vbTextCompare) > 0)
        Case "IN"
            val = Replace(val, "(", ""): val = Replace(val, ")", "")
            Dim arr(): arr = Split(val, ",")
            Dim i: For i = LBound(arr) To UBound(arr)
                If Trim$(actual) = Trim$(Replace(arr(i), """", "")) Then EvalExpr = True: Exit Function
            Next i
            EvalExpr = False
        Case Else: EvalExpr = False
    End Select
End Function
```

✅ Step 3: UserForm – frmConceptViewer

Add a UserForm named `frmConceptViewer` with:

- TreeView: `tvNav`

- ListView: `lvMeta` (View = Report)

- Label: `lblSummary`

```
Private Sub UserForm_Initialize()
    lvMeta.ColumnHeaders.Clear
    lvMeta.ColumnHeaders.Add , , "Key", 180
    lvMeta.ColumnHeaders.Add , , "Value", 320

    mConceptEngine.Build
    BuildTree
    lblSummary.Caption = CStr(mConceptEngine.Concepts.Count) & "
concepts loaded"
End Sub

Private Sub BuildTree()
    tvNav.Nodes.Clear
    Dim k
    For Each k In mConceptEngine.Concepts.Keys
        Dim c As cConcept: Set c = mConceptEngine.Concepts(k)
        If Len(c.ParentID) = 0 Then
            tvNav.Nodes.Add , , c.ID, "[" & c.Kind & "] " & c.Title
            AddChildren c.ID
        End If
    Next k
    tvNav.ExpandAll
End Sub

Private Sub AddChildren(parent$)
    If Not mConceptEngine.ParentMap.Exists(parent) Then Exit Sub
    Dim ch As Collection: Set ch = mConceptEngine.ParentMap(parent)
    Dim i&
    For i = 1 To ch.Count
        Dim cid$: cid = ch(i)
        Dim c As cConcept: Set c = mConceptEngine.Concepts(cid)
        tvNav.Nodes.Add c.ParentID, tvwChild, c.ID, "[" & c.Kind & "] "
& c.Title
        AddChildren c.ID
    Next i
End Sub

Private Sub tvNav_NodeClick(ByVal Node As MSComctlLib.Node)
    ShowMeta Node.Key
End Sub

Private Sub ShowMeta(id$)
    Dim c As cConcept: Set c = mConceptEngine.Concepts(id)
    lvMeta.ListItems.Clear
    AddRow "Title", c.Title
    AddRow "Kind", c.Kind

    Dim k
    For Each k In c.Meta.Keys
        AddRow k, c.Meta(k)
    Next k

```

```
    Dim results As Collection: Set results =
mConceptEngine.EvaluateConcept(c)
    Dim i&
    For i = 1 To results.Count
        Dim r As Scripting.Dictionary: Set r = results(i)
        AddRow "Rule " & r("RuleID"), r("Status") & " [" &
r("Severity") & "]"
        AddRow "→", r("Message")
    Next i
End Sub

Private Sub AddRow(k$, v$)
    Dim it As ListItem
    Set it = lvMeta.ListItems.Add(, , k$)
    it.SubItems(1) = v$
End Sub
```

- # How to Run

VBA logigram and algorigram for digital logic and curriculum mapping

Below is a compact, production-ready VBA package that does two things:

- Digital logic engine: parses your sum expressions (e.g., S0 = X1 + X3 + X5), resolves dependencies among S-terms, validates symbols, and generates a truth table for inputs X1…Xn.
- Curriculum logigram: maps Domains → Modules → Outcomes/Assessments and flags gaps (unmapped outcomes or modules with no assessments).

Use the exact sheet schemas shown, then paste the code.

1) Workbook sheets

Create these sheets with exact headers.

Sheet: LogicRules

- Columns: ID, Expr
- Example:
  - S0 | X1 + X3 + X5

    - S1 | X2 + X3 + X6 + X + X7
    - S3 | S4 + X5 + X6
    - S4 | X4

Notes:

- Use + for OR. Optional: use * for AND and ' for NOT (e.g., X1*X2' + X3). If you don’t use AND/NOT, it still works with OR-only.
- If a rule references an unknown symbol (like S4 missing or stray X), the validator flags it.

Sheet: LogicInputs

- Columns: Var
- List your input variables (e.g., X1, X2, X3, X4, X5, X6, X7).

Sheet: TruthTable

- Leave empty; code will populate: all input combinations (limited to ≤ 8 inputs for 256 rows) and computed S-outputs.

Sheet: Curriculum

- Columns: Domain, Module, Outcome, Assessment
- Example rows:
    - Digital Logic & Electronics | Register mapping | Derive register selects | Truth table, gate-level sim
    - Control Systems & Automation | Block diagrams | Analyze feedback loop | Block diagram analysis
    - Trade Theory & Safety | SABS wiring codes | Apply SABS codes | Inspection checklist

Sheet: CurriculumFindings

- Leave empty; code writes findings (e.g., missing outcomes, unassessed modules).

2) Class: cNode (for curriculum logigram)

```
' Class Module: cNode
Option Explicit
Public ID As String
Public ParentID As String
Public Title As String
Public Kind As String
Public Meta As Scripting.Dictionary

Private Sub Class_Initialize()
    Set Meta = New Scripting.Dictionary
End Sub
' Module: mLogic
Option Explicit

' Requires reference: Microsoft Scripting Runtime

Private Type Rule
    Name As String
    Expr As String
    RPN As Collection         ' Reverse Polish Notation tokens
    DependsOn As Scripting.Dictionary ' symbol -> True
End Type

Private Rules As Scripting.Dictionary           ' Name -> Rule
Private Inputs As Scripting.Dictionary          ' Input symbol -> True
Private Symbols As Scripting.Dictionary         ' All symbols (inputs and S)
-> "INPUT"/"DERIVED"
Private Order As Collection                     ' Topological order of S
symbols

' Public entrypoints
Public Sub BuildLogicModel()
    LoadInputs
    LoadRules
    ValidateSymbols
    BuildDependencies
    TopoSort
End Sub

Public Sub GenerateTruthTable()
    If Inputs Is Nothing Then BuildLogicModel
    Dim ws As Worksheet: Set ws = SheetByName("TruthTable", True)
    Dim inputList As Collection: Set inputList = KeysToCollection(Inputs)

    Dim n As Long: n = inputList.Count
    If n = 0 Then Err.Raise 5, , "No inputs listed in LogicInputs."
    If n > 8 Then Err.Raise 5, , "Too many inputs (" & n & "). Limit to 8 for
truth table."

    ' Header
    Dim c As Long, r As Long: r = 1: c = 1
    Dim i As Long
```

```
    For i = 1 To n
        ws.Cells(r, c).Value = CStr(inputList(i)): c = c + 1
    Next i
    Dim sNames As Collection: Set sNames = DerivedSNames()
    Dim j As Long
    For j = 1 To sNames.Count
        ws.Cells(r, c).Value = CStr(sNames(j)): c = c + 1
    Next j

    ' Rows
    Dim rowsMax As Long: rowsMax = 2 ^ n
    Dim assign As Scripting.Dictionary
    Set assign = New Scripting.Dictionary

    Dim row As Long
    For row = 0 To rowsMax - 1
        r = r + 1: c = 1
        ' set inputs
        For i = 1 To n
            Dim bit As Long: bit = (row \ (2 ^ (n - i))) And 1
            ws.Cells(r, c).Value = bit
            assign(CStr(inputList(i))) = CBool(bit)
            c = c + 1
        Next i
        ' compute S in topological order
        Dim sVal As Scripting.Dictionary: Set sVal = EvalDerived(assign)
        For j = 1 To sNames.Count
            ws.Cells(r, c).Value = IIf(sVal.Exists(CStr(sNames(j))) And
sVal(CStr(sNames(j))) = True, 1, 0)
            c = c + 1
        Next j
    Next row

    ws.Columns.AutoFit
End Sub

' ========== Internals ==========

Private Sub LoadInputs()
    Set Inputs = New Scripting.Dictionary
    Set Symbols = New Scripting.Dictionary

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("LogicInputs")
    Dim r As Long, last As Long: last = ws.Cells(ws.Rows.Count,
1).End(xlUp).Row
    For r = 2 To last
        Dim v As String: v = Trim$(CStr(ws.Cells(r, 1).Value2))
        If Len(v) > 0 Then
            Inputs(UCase$(v)) = True
            Symbols(UCase$(v)) = "INPUT"
        End If
    Next r
End Sub

Private Sub LoadRules()
```

```
    Set Rules = New Scripting.Dictionary

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("LogicRules")
    Dim r As Long, last As Long: last = ws.Cells(ws.Rows.Count,
1).End(xlUp).Row
    For r = 2 To last
        Dim name As String, expr As String
        name = Trim$(CStr(ws.Cells(r, 1).Value2))
        expr = Trim$(CStr(ws.Cells(r, 2).Value2))
        If Len(name) = 0 Or Len(expr) = 0 Then GoTo NextR
        Dim rr As Rule
        rr.Name = UCase$(name)
        rr.Expr = expr
        Set rr.RPN = InfixToRPN(expr, rr.DependsOn)
        Rules(rr.Name) = rr
        Symbols(rr.Name) = "DERIVED"
NextR:
    Next r
End Sub

Private Sub ValidateSymbols()
    ' Check that every symbol in dependencies is either input or rule
    Dim k As Variant
    For Each k In Rules.Keys
        Dim rr As Rule: rr = Rules(k)
        Dim dep As Variant
        For Each dep In rr.DependsOn.Keys
            If Not Symbols.Exists(dep) Then
                ' Unknown symbol -> warning in immediate window
                Debug.Print "Unknown symbol in expression of " & rr.Name & ":
" & dep
            End If
        Next dep
    Next k
End Sub

Private Sub BuildDependencies()
    ' Already built per rule (DependsOn)
End Sub

Private Sub TopoSort()
    ' Kahn's algorithm over derived S-terms
    Set Order = New Collection
    Dim indeg As Scripting.Dictionary: Set indeg = New Scripting.Dictionary
    Dim s As Variant
    For Each s In Rules.Keys
        indeg(s) = 0
    Next s

    ' Count dependencies among DERIVED only
    Dim k As Variant, dep As Variant
    For Each k In Rules.Keys
        Dim rr As Rule: rr = Rules(k)
        For Each dep In rr.DependsOn.Keys
            If Symbols.Exists(dep) And Symbols(dep) = "DERIVED" Then
```

```
                indeg(k) = indeg(k) + 1
            End If
        Next dep
    Next k

    ' Queue
    Dim Q As Collection: Set Q = New Collection
    For Each k In indeg.Keys
        If indeg(k) = 0 Then Q.Add k
    Next k

    Do While Q.Count > 0
        Dim n As String: n = CStr(Q(1)): Q.Remove 1
        Order.Add n
        ' Decrease neighbors (find rules that depend on n)
        For Each k In Rules.Keys
            Dim rr As Rule: rr = Rules(k)
            If rr.DependsOn.Exists(n) Then
                indeg(k) = indeg(k) - 1
                If indeg(k) = 0 Then Q.Add k
            End If
        Next k
    Loop

    ' Detect cycles
    If Order.Count < Rules.Count Then
        Debug.Print "Warning: cyclic dependency among S-terms. Evaluation may
fail."
    End If
End Sub

Private Function EvalDerived(assign As Scripting.Dictionary) As
Scripting.Dictionary
    Dim val As New Scripting.Dictionary
    Dim i As Long
    ' Set inputs as values
    Dim k As Variant
    For Each k In assign.Keys
        val(UCase$(CStr(k))) = CBool(assign(k))
    Next k

    ' Evaluate in topological order
    For i = 1 To Order.Count
        Dim sName As String: sName = CStr(Order(i))
        Dim rr As Rule: rr = Rules(sName)
        val(sName) = EvalRPN(rr.RPN, val)
    Next i
    Set EvalDerived = val
End Function

Private Function DerivedSNames() As Collection
    Dim c As New Collection, k As Variant
    For Each k In Order
        c.Add CStr(k)
    Next k
```

```
    Set DerivedSNames = c
End Function

' ======= Expression parsing: Infix to RPN (Shunting-yard) =======
' Supported:
'   +  OR
'   *  AND (optional)
'   '  NOT (postfix, e.g., X1' ; optional)
'   parentheses ( )
'   symbols: [A-Za-z][A-Za-z0-9_]*

Private Function InfixToRPN(expr As String, ByRef deps As
Scripting.Dictionary) As Collection
    Dim toks As Collection: Set toks = Tokenize(expr)
    Dim outQ As New Collection, opStk As New Collection
    Dim i As Long
    Set deps = New Scripting.Dictionary

    For i = 1 To toks.Count
        Dim t As String: t = toks(i)
        If IsSymbol(t) Then
            outQ.Add UCase$(t)
            deps(UCase$(t)) = True
        ElseIf t = "'" Then
            ' postfix NOT applies to previous output token
            outQ.Add "'"
        ElseIf t = "+" Or t = "*" Then
            Do While opStk.Count > 0 AndAlso
Precedence(CStr(opStk(opStk.Count))) >= Precedence(t)
                outQ.Add opStk(opStk.Count): opStk.Remove opStk.Count
            Loop
            opStk.Add t
        ElseIf t = "(" Then
            opStk.Add t
        ElseIf t = ")" Then
            Do While opStk.Count > 0 And CStr(opStk(opStk.Count)) <> "("
                outQ.Add opStk(opStk.Count): opStk.Remove opStk.Count
            Loop
            If opStk.Count > 0 And CStr(opStk(opStk.Count)) = "(" Then
opStk.Remove opStk.Count
        End If
    Next i

    Do While opStk.Count > 0
        outQ.Add opStk(opStk.Count): opStk.Remove opStk.Count
    Loop

    Set InfixToRPN = outQ
End Function

Private Function Tokenize(ByVal s As String) As Collection
    Dim c As New Collection, i As Long, cur As String, ch As String
    For i = 1 To Len(s)
        ch = Mid$(s, i, 1)
        Select Case ch
```

```
            Case " ", vbTab
                If Len(cur) > 0 Then c.Add cur: cur = ""
            Case "+", "*", "(", ")", "'"
                If Len(cur) > 0 Then c.Add cur: cur = ""
                c.Add ch
            Case Else
                cur = cur & ch
        End Select
    Next i
    If Len(cur) > 0 Then c.Add cur
    Set Tokenize = c
End Function

Private Function IsSymbol(t As String) As Boolean
    Dim ch As String: ch = Left$(t, 1)
    IsSymbol = (ch Like "[A-Za-z]")
End Function

Private Function Precedence(op As String) As Long
    Select Case op
        Case "'": Precedence = 3
        Case "*": Precedence = 2
        Case "+": Precedence = 1
        Case Else: Precedence = 0
    End Select
End Function

Private Function EvalRPN(rpn As Collection, vals As Scripting.Dictionary) As
Boolean
    Dim st As New Collection, i As Long
    For i = 1 To rpn.Count
        Dim t As String: t = rpn(i)
        If IsSymbol(t) Then
            st.Add BoolVal(vals, t)
        ElseIf t = "'" Then
            Dim a As Boolean: a = PopBool(st): st.Add Not a
        ElseIf t = "+" Or t = "*" Then
            Dim b As Boolean: b = PopBool(st)
            Dim a As Boolean: a = PopBool(st)
            If t = "+" Then st.Add (a Or b) Else st.Add (a And b)
        Else
            ' ignore
        End If
    Next i
    EvalRPN = PopBool(st)
End Function

Private Function PopBool(st As Collection) As Boolean
    If st.Count = 0 Then PopBool = False: Exit Function
    PopBool = CBool(st(st.Count))
    st.Remove st.Count
End Function

Private Function BoolVal(vals As Scripting.Dictionary, key As String) As
Boolean
```

```
    If vals.Exists(key) Then
        BoolVal = CBool(vals(key))
    Else
        ' Unknown symbol -> assume False but log
        Debug.Print "Value for symbol not set: "; key
        BoolVal = False
    End If
End Function

Private Function KeysToCollection(d As Scripting.Dictionary) As Collection
    Dim c As New Collection, k As Variant
    For Each k In d.Keys: c.Add k: Next k
    Set KeysToCollection = c
End Function

Private Function SheetByName(name As String, Optional createIfMissing As
Boolean = False) As Worksheet
    On Error Resume Next
    Set SheetByName = ThisWorkbook.Worksheets(name)
    On Error GoTo 0
    If SheetByName Is Nothing And createIfMissing Then
        Set SheetByName = ThisWorkbook.Worksheets.Add
        SheetByName.Name = name
    End If
End Function

Private Function AndAlso(a As Boolean, b As Boolean) As Boolean
    If a Then AndAlso = b Else AndAlso = False
End Function
' Module: mCurriculum
Option Explicit

' Requires: Microsoft Scripting Runtime, cNode class

Public Nodes As Scripting.Dictionary
Public ParentMap As Scripting.Dictionary

Public Sub BuildCurriculumLogigram()
    Set Nodes = New Scripting.Dictionary
    Set ParentMap = New Scripting.Dictionary

    Dim ws As Worksheet: Set ws = ThisWorkbook.Worksheets("Curriculum")
    Dim r As Long, last As Long: last = ws.Cells(ws.Rows.Count,
1).End(xlUp).Row

    Dim dom$, modl$, outc$, assess$
    For r = 2 To last
        dom = NzStr(ws.Cells(r, 1).Value2)
        modl = NzStr(ws.Cells(r, 2).Value2)
        outc = NzStr(ws.Cells(r, 3).Value2)
        assess = NzStr(ws.Cells(r, 4).Value2)

        If Len(dom) > 0 Then EnsureNode "D_" & Key(dom), "", dom, "Domain"
        If Len(modl) > 0 Then EnsureNode "M_" & Key(dom & "|" & modl), "D_" &
Key(dom), modl, "Module"
```

```
        If Len(outc) > 0 Then
            EnsureNode "O_" & Key(dom & "|" & modl & "|" & outc), "M_" &
Key(dom & "|" & modl), outc, "Outcome"
        End If
        If Len(assess) > 0 Then
            EnsureNode "A_" & Key(dom & "|" & modl & "|" & outc & "|" &
assess), IIf(Len(outc) > 0, "O_" & Key(dom & "|" & modl & "|" & outc), "M_" &
Key(dom & "|" & modl)), assess, "Assessment"
        End If
    Next r
End Sub

Public Sub CurriculumFindings()
    Dim ws As Worksheet: Set ws = SheetByName("CurriculumFindings", True)
    ws.Cells.Clear
    ws.Range("A1:D1").Value = Array("Level", "Node", "Issue", "Detail")
    Dim row As Long: row = 1

    ' Find modules with no outcomes
    Dim k As Variant
    For Each k In Nodes.Keys
        Dim n As cNode: Set n = Nodes(k)
        If n.Kind = "Module" Then
            If Not HasChildren(k) Then
                row = row + 1
                ws.Cells(row, 1).Value = "Module"
                ws.Cells(row, 2).Value = n.Title
                ws.Cells(row, 3).Value = "No outcomes/assessments"
                ws.Cells(row, 4).Value = ""
            End If
        End If
    Next k

    ' Outcomes with no assessment
    For Each k In Nodes.Keys
        Dim n2 As cNode: Set n2 = Nodes(k)
        If n2.Kind = "Outcome" Then
            If Not HasChildren(k) Then
                row = row + 1
                ws.Cells(row, 1).Value = "Outcome"
                ws.Cells(row, 2).Value = n2.Title
                ws.Cells(row, 3).Value = "No assessment"
                ws.Cells(row, 4).Value = ""
            End If
        End If
    Next k

    ws.Columns.AutoFit
End Sub

' ===== Helpers =====
Private Sub EnsureNode(id$, parent$, title$, kind$)
    If Not Nodes.Exists(id) Then
        Dim n As New cNode
        n.ID = id: n.ParentID = parent: n.Title = title: n.Kind = kind
```

```
        Nodes(id) = n
        If Len(parent) > 0 Then AddChild parent, id
    End If
End Sub

Private Sub AddChild(parent$, child$)
    If ParentMap Is Nothing Then Set ParentMap = New Scripting.Dictionary
    If Not ParentMap.Exists(parent$) Then
        Dim c As New Collection: Set ParentMap(parent$) = c
    End If
    ParentMap(parent$).Add child$
End Sub

Private Function HasChildren(id$) As Boolean
    HasChildren = (ParentMap.Exists(id$) And ParentMap(id$).Count > 0)
End Function

Private Function Key(s$) As String
    Dim t$: t = UCase$(s)
    t = Replace(t, " ", "_")
    t = Replace(t, "-", "_")
    t = Replace(t, "/", "_")
    Key = t
End Function

Private Function NzStr(v As Variant) As String
    If IsError(v) Or IsEmpty(v) Then NzStr = "" Else NzStr = CStr(v)
End Function

Private Function SheetByName(name As String, Optional createIfMissing As
Boolean = False) As Worksheet
    On Error Resume Next
    Set SheetByName = ThisWorkbook.Worksheets(name)
    On Error GoTo 0
    If SheetByName Is Nothing And createIfMissing Then
        Set SheetByName = ThisWorkbook.Worksheets.Add
        SheetByName.Name = name
    End If
End Function
' Module: mRun
Option Explicit

Public Sub Run_Logic_Validate_And_TruthTable()
    mLogic.BuildLogicModel
    mLogic.GenerateTruthTable
    MsgBox "Truth table generated.", vbInformation
End Sub

Public Sub Run_Curriculum_Logigram_And_Findings()
    mCurriculum.BuildCurriculumLogigram
    mCurriculum.CurriculumFindings
    MsgBox "Curriculum findings generated.", vbInformation
End Sub
```

6) Notes and extensions

- Unknown symbols (like stray “X” in S1, or missing S4) are reported in the Immediate Window (Ctrl+G). You can harden by writing them to a “LogicFindings” sheet if desired.
- Operators supported: + (OR), * (AND), postfix ' (NOT), and parentheses. If you only use OR, keep using +.
- To keep truth tables readable, inputs are limited to 8 (256 rows). For more, test with random vectors instead of full truth table.

If you want a viewer UserForm with a TreeView for curriculum nodes and a rule tab for logic validation, I can add it in a single drop-in form.

VBA logigram and algorigram for banking automation, circuit design, and SARB alignment

This drop-in Excel VBA builds:

- A logigram of domains: Digital Banking, Cash Management, Banknote Processing, Security Systems, Circuit Design Strategies (Component→Function), SARB Applications, Educational Alignment, Career Outcomes, Project Takeaways.
- An algorigram of checks: completeness of each domain, component/function gaps, consistency of qualification levels, presence of SARB-relevant applications.
- A Findings sheet and a Dashboard.

Workbook sheets to create

Paste your data into these sheets with exact headers.

Sheet: Domains

- Columns: Domain, Item
- Example rows:
    - Digital Banking | Mobile apps, online banking, debit/credit systems
    - Cash Management | Remote deposit capture, merchant services
    - Banknote Processing | JL 305 Series sorting machines, Linux OS, barcode reading
    - Security Systems | Counterfeit detection, blacklist comparison, traceability

Sheet: CircuitDesign

- Columns: Component, Function
- Example rows:
    - Capacitor & Resistor | Regulate flow and store charge
    - PCB Ground Plan | Prevent electromagnetic interference, improve signal integrity
    - Logic Gates | Control flow and decision-making in digital circuits
    - Power Supplies | Manage voltage and current across components
    - Joystick Switches | Convert motion into electrical signals
    - Battery Systems | Calculate discharge time and energy efficiency

Sheet: SARB_Applications

- Columns: Area, Description
- Example rows:
    - Currency Management | Banknote printing, sorting, and validation

    - ATM Systems | Diagnostics, maintenance, and circuit integration
    - Financial Analytics | Data modeling, econometrics, and forecasting
    - Security & Compliance | Health, safety, and regulatory adherence

Sheet: EducationAlignment

- Columns: Qualification Level, Description
- Example rows:
    - NQF Level 4–6 | Electrical and Electronics Engineering (N4–N6)
    - Postgraduate | Data Science, Applied Mathematics, Econometrics
    - Certifications | Python, R, GitHub contributions, SARB academic modules

Sheet: CareerOutcomes

- Columns: Role, Description
- Example rows:
    - Graduate Intern | SARB Business Solutions & Technology
    - Electronics Engineer | Circuit design, diagnostics, ATM systems
    - Data Scientist | Central banking analytics, monetary policy modeling
    - Digital Banking Specialist | Mobile platforms, customer interface systems
    - Financial Systems Developer | Currency management, fraud detection

Sheet: ProjectTakeaways

- Columns: Takeaway

- Example rows:
    - Engineering electrical and electronics are foundational to financial systems
    - Data science enhances decision-making and predictive modeling in banking
    - SARB offers a structured pathway for graduates to develop technical and analytical skills
    - Circuit design and diagnostics are critical for ATM, banknote, and digital banking systems
    - Integration of electronics, coding, and analytics is key to 4IR transformation in finance

Leave these blank; code will create them:

- Findings
- Dashboard

VBA code (paste into a standard module, e.g., mBankingFramework)

```
Option Explicit

Private gFindRow As Long

Public Sub Run_Banking_Framework_Audit()
    Application.ScreenUpdating = False
    InitOutputs

    ValidateDomains
    ValidateCircuitDesign
    ValidateSARBApplications
    ValidateEducationAlignment
    ValidateCareerOutcomes
    ValidateProjectTakeaways

    BuildDashboard

    Application.ScreenUpdating = True
    MsgBox "Audit complete. See 'Findings' and 'Dashboard'.", vbInformation
End Sub

' ========= Outputs =========
```

```
Private Sub InitOutputs()
    On Error Resume Next
    Worksheets("Findings").Delete
    Worksheets("Dashboard").Delete
    On Error GoTo 0

    Dim f As Worksheet
    Set f = Worksheets.Add(After:=Worksheets(Worksheets.Count))
    f.Name = "Findings"
    f.Range("A1:E1").Value = Array("Area", "Item", "Issue", "Detail",
"Action")
    gFindRow = 1
End Sub

Private Sub AddFinding(area$, item$, issue$, detail$, action$)
    gFindRow = gFindRow + 1
    With Worksheets("Findings")
        .Cells(gFindRow, 1).Value = area
        .Cells(gFindRow, 2).Value = item
        .Cells(gFindRow, 3).Value = issue
        .Cells[gFindRow, 4].Value = detail
        .Cells(gFindRow, 5).Value = action
    End With
End Sub

Private Function TrySheet(name$, ByRef ws As Worksheet) As Boolean
    On Error Resume Next
    Set ws = Worksheets(name)
    On Error GoTo 0
    TrySheet = Not ws Is Nothing
End Function

Private Function CountRows(sheetName$) As Long
    Dim ws As Worksheet
    If Not TrySheet(sheetName, ws) Then Exit Function
    CountRows = Application.Max(0, ws.Cells(ws.Rows.Count, 1).End(xlUp).Row -
1)
End Function

' ========= Domains (Digital Banking, Cash Mgmt, Banknote Processing,
Security) =========

Private Sub ValidateDomains()
    Dim ws As Worksheet
    If Not TrySheet("Domains", ws) Then
        AddFinding "Domains", "(Sheet)", "Missing", "Domains", "Create sheet
with Domain, Item"
        Exit Sub
    End If

    Dim req As Variant
    req = Array("Digital Banking", "Cash Management", "Banknote Processing",
"Security Systems")
```

```
    Dim i&, found As Object: Set found = CreateObject("Scripting.Dictionary")
    For i = LBound(req) To UBound(req)
        found(req(i)) = False
    Next i

    Dim r&, lastR&: lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        Dim dom$, it$
        dom = Trim$(ws.Cells(r, 1).Value)
        it = Trim$(ws.Cells(r, 2).Value)
        If Len(dom) = 0 And Len(it) = 0 Then GoTo NextR
        If Len(dom) = 0 Then AddFinding "Domains", "(Row " & r & ")",
"Missing Domain", "", "Enter domain name"
        If Len(it) = 0 Then AddFinding "Domains", dom, "Missing Item", "",
"Provide description/examples")
        If found.Exists(dom) And Len(it) > 0 Then found(dom) = True
NextR:
    Next r

    For i = LBound(req) To UBound(req)
        If Not found(req(i)) Then AddFinding "Domains", req(i), "Not
covered", "", "Add at least one item for this domain"
    Next i
End Sub

' ========= Circuit design (Component → Function) =========

Private Sub ValidateCircuitDesign()
    Dim ws As Worksheet
    If Not TrySheet("CircuitDesign", ws) Then
        AddFinding "CircuitDesign", "(Sheet)", "Missing", "CircuitDesign",
"Create sheet with Component, Function"
        Exit Sub
    End If

    Dim r&, lastR&: lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    Dim seen As Object: Set seen = CreateObject("Scripting.Dictionary")
    Dim must As Variant
    must = Array("Capacitor & Resistor", "PCB Ground Plan", "Logic Gates",
"Power Supplies", "Joystick Switches", "Battery Systems")

    Dim i&
    For i = LBound(must) To UBound(must)
        seen(must(i)) = False
    Next i

    For r = 2 To lastR
        Dim comp$, func$
        comp = Trim$(ws.Cells(r, 1).Value)
        func = Trim$(ws.Cells(r, 2).Value)
        If Len(comp) = 0 And Len(func) = 0 Then GoTo NextR
        If Len(comp) = 0 Then AddFinding "CircuitDesign", "(Row " & r & ")",
"Missing component", "", "Enter component name"
        If Len(func) = 0 Then AddFinding "CircuitDesign", comp, "Missing
function", "", "Describe purpose/role"
```

```
        If seen.Exists(comp) And Len(func) > 0 Then seen(comp) = True
NextR:
    Next r

    For i = LBound(must) To UBound(must)
        If Not seen(must(i)) Then AddFinding "CircuitDesign", must(i), "Not
found", "", "Add this component row"
    Next i
End Sub

' ========= SARB Applications =========

Private Sub ValidateSARBApplications()
    Dim ws As Worksheet
    If Not TrySheet("SARB_Applications", ws) Then
        AddFinding "SARB_Applications", "(Sheet)", "Missing",
"SARB_Applications", "Create sheet with Area, Description"
        Exit Sub
    End If

    Dim required As Variant
    required = Array("Currency Management", "ATM Systems", "Financial
Analytics", "Security & Compliance")

    Dim present As Object: Set present = CreateObject("Scripting.Dictionary")
    Dim i&
    For i = LBound(required) To UBound(required)
        present(required(i)) = False
    Next i

    Dim r&, lastR&: lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        Dim area$, desc$
        area = Trim$(ws.Cells(r, 1).Value)
        desc = Trim$(ws.Cells(r, 2).Value)
        If Len(area) = 0 And Len(desc) = 0 Then GoTo NextR
        If Len(desc) = 0 Then AddFinding "SARB_Applications", area, "Missing
description", "", "Provide scope or examples"
        If present.Exists(area) And Len(desc) > 0 Then present(area) = True
NextR:
    Next r

    For i = LBound(required) To UBound(required)
        If Not present(required(i)) Then AddFinding "SARB_Applications",
required(i), "Not covered", "", "Add this application area"
    Next i
End Sub

' ========= Education alignment =========

Private Sub ValidateEducationAlignment()
    Dim ws As Worksheet
    If Not TrySheet("EducationAlignment", ws) Then
        AddFinding "EducationAlignment", "(Sheet)", "Missing",
"EducationAlignment", "Create sheet with Qualification Level, Description"
```

```
        Exit Sub
    End If

    Dim haveNQF As Boolean, havePG As Boolean, haveCert As Boolean
    Dim r&, lastR&: lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        Dim lvl$, desc$
        lvl = UCase$(Trim$(ws.Cells(r, 1).Value))
        desc = Trim$(ws.Cells(r, 2).Value)
        If Len(lvl) = 0 And Len(desc) = 0 Then GoTo NextR
        If Len(desc) = 0 Then AddFinding "EducationAlignment", lvl, "Missing
description", "", "Add summary/curriculum context"
        haveNQF = haveNQF Or (InStr(lvl, "NQF") > 0 Or InStr(lvl, "N4") > 0
Or InStr(lvl, "N5") > 0 Or InStr(lvl, "N6") > 0)
        havePG = havePG Or (InStr(lvl, "POSTGRADUATE") > 0)
        haveCert = haveCert Or (InStr(lvl, "CERT") > 0)
NextR:
    Next r

    If Not haveNQF Then AddFinding "EducationAlignment", "NQF Level 4–6",
"Missing", "", "Add N-level context (N4–N6)"
    If Not havePG Then AddFinding "EducationAlignment", "Postgraduate",
"Missing", "", "Add PG pathways (Data Science/Econometrics)"
    If Not haveCert Then AddFinding "EducationAlignment", "Certifications",
"Missing", "", "List Python/R/GitHub/SARB modules"
End Sub

' ========= Career outcomes =========

Private Sub ValidateCareerOutcomes()
    Dim ws As Worksheet
    If Not TrySheet("CareerOutcomes", ws) Then
        AddFinding "CareerOutcomes", "(Sheet)", "Missing", "CareerOutcomes",
"Create sheet with Role, Description"
        Exit Sub
    End If

    Dim r&, lastR&: lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    Dim need As Variant
    need = Array("Graduate Intern", "Electronics Engineer", "Data Scientist",
"Digital Banking Specialist", "Financial Systems Developer")

    Dim present As Object: Set present = CreateObject("Scripting.Dictionary")
    Dim i&
    For i = LBound(need) To UBound(need)
        present(need(i)) = False
    Next i

    For r = 2 To lastR
        Dim role$, desc$
        role = Trim$(ws.Cells(r, 1).Value)
        desc = Trim$(ws.Cells(r, 2).Value)
        If Len(role) = 0 And Len(desc) = 0 Then GoTo NextR
        If Len(desc) = 0 Then AddFinding "CareerOutcomes", role, "Missing
description", "", "Add key duties/skills")
```

```
        If present.Exists(role) And Len(desc) > 0 Then present(role) = True
NextR:
    Next r

    For i = LBound(need) To UBound(need)
        If Not present(need(i)) Then AddFinding "CareerOutcomes", need(i),
"Not covered", "", "Add role row"
    Next i
End Sub

' ========= Project takeaways =========

Private Sub ValidateProjectTakeaways()
    Dim ws As Worksheet
    If Not TrySheet("ProjectTakeaways", ws) Then
        AddFinding "ProjectTakeaways", "(Sheet)", "Missing",
"ProjectTakeaways", "Create sheet with Takeaway"
        Exit Sub
    End If

    Dim r&, lastR&, count&
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        If Len(Trim$(ws.Cells(r, 1).Value)) > 0 Then count = count + 1
    Next r
    If count = 0 Then AddFinding "ProjectTakeaways", "All", "No takeaways
listed", "", "Add 3–5 concise bullets")
End Sub

' ========= Dashboard =========

Private Sub BuildDashboard()
    Dim ws As Worksheet
    Set ws = Worksheets.Add(After:=Worksheets(Worksheets.Count))
    ws.Name = "Dashboard"
    ws.Range("A1:D1").Value = Array("Metric", "Value", "Notes", "Source")

    Dim r&: r = 1

    r = r + 1: ws.Cells(r, 1).Value = "Domains covered"
    ws.Cells(r, 2).Value = CountRows("Domains")
    ws.Cells(r, 4).Value = "Domains"

    r = r + 1: ws.Cells(r, 1).Value = "Circuit components mapped"
    ws.Cells(r, 2).Value = CountRows("CircuitDesign")
    ws.Cells(r, 4).Value = "CircuitDesign"

    r = r + 1: ws.Cells(r, 1).Value = "SARB application areas"
    ws.Cells(r, 2).Value = CountRows("SARB_Applications")
    ws.Cells(r, 4).Value = "SARB_Applications"

    r = r + 1: ws.Cells(r, 1).Value = "Qualification items"
    ws.Cells(r, 2).Value = CountRows("EducationAlignment")
    ws.Cells(r, 4).Value = "EducationAlignment"
```

```
    r = r + 1: ws.Cells(r, 1).Value = "Career outcomes"
    ws.Cells(r, 2).Value = CountRows("CareerOutcomes")
    ws.Cells(r, 4).Value = "CareerOutcomes"

    r = r + 1: ws.Cells(r, 1).Value = "Project takeaways"
    ws.Cells(r, 2).Value = CountRows("ProjectTakeaways")
    ws.Cells(r, 4).Value = "ProjectTakeaways"

    ws.Columns.AutoFit
End Sub
```

How to run

1. Create the sheets listed above and paste your content.
2. Alt+F11 → Insert → Module → paste the code.
3. Run: Run_Banking_Framework_Audit.
4. Review “Findings” (gaps, actions) and “Dashboard” (roll-up metrics).

Optional extensions

- Add a “CapabilitiesMatrix” sheet to map each domain item to required skills (electronics, software, compliance), and extend the validator to flag unassigned skills.
- Add a PoE_Checklist generator that turns SARB_Applications + CircuitDesign into evidence lines for a portfolio or Expo submission.
- Add a printable “Portfolio” sheet merging top findings, key roles, and a one-page summary for SARB applications.

his drop-in Excel VBA turns your multi-sector plan into:

- A logigram of domains (career alignment, process planning, QC, materials, electrical power, software/logic, quals, specs, networking, security, records, git/azure, compliance, computer systems, math/physics, career DB, PC architecture, integrations).
- An algorigram of checks for completeness, consistency, and readiness.
- Findings and Dashboard sheets for audit and portfolio packaging.

Workbook sheets to create

Create these sheets with exact headers (paste your content as rows).

- CareerAlignment: Role | Skills Required
- ProcessPlanning: Process Type | Techniques Included
- InspectionQC: Topic | Detail
- MaterialsScience: Category | Items
- ElectricalInstall: Topic | Detail
- SoftwareLogic: Topic | Detail
- IntlQualAlign: Country | Qualification Framework | Alignment Notes
- ProjectOutcomes: Outcome
- ProductSpecs: Component | Description
- NetworkingToolkits: Topic | Detail
- DigitalSecurity: Feature | Description
- EducationRecords: Area | Description
- GitIntegration: Topic | Detail
- IrregularityCompliance: Entity | Role
- ComputerSystems: Component | Description
- MathPhysics: Topic | FormulaOrConcept
- CareerDocsDB: Document | Description
- PCArchitecture: Component | Description
- IntegrationApps: Topic | Description

Leave blank (code creates them): Findings, Dashboard.

VBA code (paste into a standard module, e.g., mNatFramework)

vba

```
Option Explicit

' Findings row tracker
Private gFindRow As Long

Public Sub Run_National_Framework_Audit()
    Application.ScreenUpdating = False
    InitOutputs

    ValidateCareerAlignment
    ValidateProcessPlanning
    ValidateInspectionQC
    ValidateMaterialsScience
    ValidateElectricalInstall
    ValidateSoftwareLogic
    ValidateIntlQualAlign
    ValidateProjectOutcomes
    ValidateProductSpecs
    ValidateNetworkingToolkits
    ValidateDigitalSecurity
    ValidateEducationRecords
    ValidateGitIntegration
    ValidateIrregularityCompliance
    ValidateComputerSystems
    ValidateMathPhysics
    ValidateCareerDocsDB
    ValidatePCArchitecture
    ValidateIntegrationApps

    BuildDashboard

    Application.ScreenUpdating = True
    MsgBox "Audit complete. See 'Findings' and 'Dashboard'.", vbInformation
End Sub

' ========= Outputs =========

Private Sub InitOutputs()
    On Error Resume Next
    Worksheets("Findings").Delete
    Worksheets("Dashboard").Delete
    On Error GoTo 0

    Dim f As Worksheet
    Set f = Worksheets.Add(After:=Worksheets(Worksheets.Count))
    f.Name = "Findings"
    f.Range("A1:E1").Value = Array("Area", "Item", "Issue", "Detail",
"Action")
    gFindRow = 1
End Sub

Private Sub AddFinding(area$, item$, issue$, detail$, action$)
    gFindRow = gFindRow + 1
```

```
    With Worksheets("Findings")
        .Cells(gFindRow, 1).Value = area
        .Cells(gFindRow, 2).Value = item
        .Cells(gFindRow, 3).Value = issue
        .Cells(gFindRow, 4).Value = detail
        .Cells(gFindRow, 5).Value = action
    End With
End Sub

Private Function TrySheet(name$, ByRef ws As Worksheet) As Boolean
    On Error Resume Next
    Set ws = Worksheets(name)
    On Error GoTo 0
    TrySheet = Not ws Is Nothing
End Function

Private Function CountRows(sheetName$) As Long
    Dim ws As Worksheet
    If Not TrySheet(sheetName, ws) Then Exit Function
    CountRows = Application.Max(0, ws.Cells(ws.Rows.Count, 1).End(xlUp).Row -
1)
End Function

' ========= 1) Career Alignment =========
Private Sub ValidateCareerAlignment()
    Dim ws As Worksheet
    If Not TrySheet("CareerAlignment", ws) Then
        AddFinding "CareerAlignment", "(Sheet)", "Missing",
"CareerAlignment", "Create Role, Skills Required"
        Exit Sub
    End If
    Dim need As Variant: need = Array("Electronics Engineer", "Software
Developer", "Data Scientist", "Banking Technologist")
    RequireNamedRows ws, 1, need, "Role", "CareerAlignment"
    RequireNonEmptySecond ws, "Skills Required", "CareerAlignment"
End Sub

' ========= 2) Process Planning =========
Private Sub ValidateProcessPlanning()
    Dim ws As Worksheet
    If Not TrySheet("ProcessPlanning", ws) Then
        AddFinding "ProcessPlanning", "(Sheet)", "Missing",
"ProcessPlanning", "Create Process Type, Techniques Included"
        Exit Sub
    End If
    Dim need As Variant: need = Array("Primary", "Secondary", "Cold Working",
"Joining", "Surface Finishing")
    RequireNamedRows ws, 1, need, "Process Type", "ProcessPlanning"
    RequireNonEmptySecond ws, "Techniques Included", "ProcessPlanning"
End Sub

' ========= 3) Inspection & QC =========
Private Sub ValidateInspectionQC()
    Dim ws As Worksheet
    If Not TrySheet("InspectionQC", ws) Then
```

```
        AddFinding "InspectionQC", "(Sheet)", "Missing", "InspectionQC",
"Create Topic, Detail"
        Exit Sub
    End If
    Dim must As Variant: must = Array("Dimensional analysis", "Control
charts", "Surface finish", "Fit types", "Tools")
    RequireTopicPresence ws, must, "InspectionQC"
End Sub

' ========= 4) Materials Science =========
Private Sub ValidateMaterialsScience()
    Dim ws As Worksheet
    If Not TrySheet("MaterialsScience", ws) Then
        AddFinding "MaterialsScience", "(Sheet)", "Missing",
"MaterialsScience", "Create Category, Items"
        Exit Sub
    End If
    Dim need As Variant: need = Array("Ferrous", "Non-ferrous", "Iron ores",
"Steel grades")
    RequireNamedRows ws, 1, need, "Category", "MaterialsScience"
    RequireNonEmptySecond ws, "Items", "MaterialsScience"
End Sub

' ========= 5) Electrical Installation & Power =========
Private Sub ValidateElectricalInstall()
    Dim ws As Worksheet
    If Not TrySheet("ElectricalInstall", ws) Then
        AddFinding "ElectricalInstall", "(Sheet)", "Missing",
"ElectricalInstall", "Create Topic, Detail"
        Exit Sub
    End If
    ' Check standards, power factor, substation design
    RequireTopicPresence ws, Array("IEC 60364", "Power factor correction",
"MV/LV substation", "Fault current"), "ElectricalInstall"
    ' Formula presence checks (as text)
    RequireDetailPattern ws, "Fault current", "Uo", "Zs", "I_d = U_o / Z_s",
"Add Id = Uo/Zs text/equation"
    RequireAnyPattern ws, Array("I = 150", "I = 150×1000"),
"ElectricalInstall", "Current calc example missing", "Add I =
150×1000/(400×√3)"
End Sub

' ========= 6) Software Engineering & Digital Logic =========
Private Sub ValidateSoftwareLogic()
    Dim ws As Worksheet
    If Not TrySheet("SoftwareLogic", ws) Then
        AddFinding "SoftwareLogic", "(Sheet)", "Missing", "SoftwareLogic",
"Create Topic, Detail"
        Exit Sub
    End If
    RequireTopicPresence ws, Array("Flowcharts", "Boolean logic",
"Hexadecimal", "Truth tables", "Sequential logic"), "SoftwareLogic"
End Sub

' ========= 7) International Qualification Alignment =========
```

```
Private Sub ValidateIntlQualAlign()
    Dim ws As Worksheet
    If Not TrySheet("IntlQualAlign", ws) Then
        AddFinding "IntlQualAlign", "(Sheet)", "Missing", "IntlQualAlign",
"Create Country, Qualification Framework, Alignment Notes"
        Exit Sub
    End If
    RequireCountry ws, "South Africa"
    RequireCountry ws, "Scotland"
    RequireAlignmentDetail ws
End Sub

' ========= 8) Project Outcomes (summary list) =========
Private Sub ValidateProjectOutcomes()
    Dim ws As Worksheet
    If Not TrySheet("ProjectOutcomes", ws) Then
        AddFinding "ProjectOutcomes", "(Sheet)", "Missing",
"ProjectOutcomes", "Create Outcome"
        Exit Sub
    End If
    If CountRows("ProjectOutcomes") < 3 Then
        AddFinding "ProjectOutcomes", "Coverage", "Too few outcomes",
CStr(CountRows("ProjectOutcomes")), "List 3–5 key outcomes"
    End If
End Sub

' ========= 9) Product Specifications =========
Private Sub ValidateProductSpecs()
    Dim ws As Worksheet
    If Not TrySheet("ProductSpecs", ws) Then
        AddFinding "ProductSpecs", "(Sheet)", "Missing", "ProductSpecs",
"Create Component, Description"
        Exit Sub
    End If
    RequireTopicPresence ws, Array("LCD Monitor", "Case Type", "Power
Supply", "UPS Systems", "Patch Panels"), "ProductSpecs"
End Sub

' ========= 10) Networking & Toolkits =========
Private Sub ValidateNetworkingToolkits()
    Dim ws As Worksheet
    If Not TrySheet("NetworkingToolkits", ws) Then
        AddFinding "NetworkingToolkits", "(Sheet)", "Missing",
"NetworkingToolkits", "Create Topic, Detail"
        Exit Sub
    End If
    RequireTopicPresence ws, Array("Cabling", "Toolkits", "Connectors",
"Testing Devices"), "NetworkingToolkits"
End Sub

' ========= 11) Digital Security & Data Management =========
Private Sub ValidateDigitalSecurity()
    Dim ws As Worksheet
    If Not TrySheet("DigitalSecurity", ws) Then
```

```
        AddFinding "DigitalSecurity", "(Sheet)", "Missing",
"DigitalSecurity", "Create Feature, Description"
        Exit Sub
    End If
    RequireTopicPresence ws, Array("Antivirus Engine", "Data Protection",
"Client Management", "Database Systems"), "DigitalSecurity"
End Sub

' ========= 12) Education & Graduation Records =========
Private Sub ValidateEducationRecords()
    Dim ws As Worksheet
    If Not TrySheet("EducationRecords", ws) Then
        AddFinding "EducationRecords", "(Sheet)", "Missing",
"EducationRecords", "Create Area, Description"
        Exit Sub
    End If
    RequireTopicPresence ws, Array("Graduation", "Career Records",
"Orientation", "Projection"), "EducationRecords"
End Sub

' ========= 13) GitLab / GitHub / Azure =========
Private Sub ValidateGitIntegration()
    Dim ws As Worksheet
    If Not TrySheet("GitIntegration", ws) Then
        AddFinding "GitIntegration", "(Sheet)", "Missing", "GitIntegration",
"Create Topic, Detail"
        Exit Sub
    End If
    RequireTopicPresence ws, Array("Triggered Projects", "Issue Management",
"Contribution Logs", "Platform Integration"), "GitIntegration"
End Sub

' ========= 14) Irregularity Management & Compliance =========
Private Sub ValidateIrregularityCompliance()
    Dim ws As Worksheet
    If Not TrySheet("IrregularityCompliance", ws) Then
        AddFinding "IrregularityCompliance", "(Sheet)", "Missing",
"IrregularityCompliance", "Create Entity, Role"
        Exit Sub
    End If
    RequireTopicPresence ws, Array("DBE", "DHET", "Umalusi"),
"IrregularityCompliance"
End Sub

' ========= 15) Computer Systems & Digital Electronics =========
Private Sub ValidateComputerSystems()
    Dim ws As Worksheet
    If Not TrySheet("ComputerSystems", ws) Then
        AddFinding "ComputerSystems", "(Sheet)", "Missing",
"ComputerSystems", "Create Component, Description"
        Exit Sub
    End If
    RequireTopicPresence ws, Array("Input Devices", "Memory Systems",
"Storage", "Logic Circuits", "Digital Processing"), "ComputerSystems"
End Sub
```

```
' ========= 16) Engineering Mathematics & Physics =========
Private Sub ValidateMathPhysics()
    Dim ws As Worksheet
    If Not TrySheet("MathPhysics", ws) Then
        AddFinding "MathPhysics", "(Sheet)", "Missing", "MathPhysics",
"Create Topic, FormulaOrConcept"
        Exit Sub
    End If
    RequireTopicPresence ws, Array("Geometry", "Integration", "Volume", "Heat
transfer", "Electrostatics", "DC/AC motor"), "MathPhysics"
    RequireAnyPattern ws, Array("V = πr^2 h", "V=πr2h", "pi r^2 h"),
"MathPhysics", "Cylinder volume formula missing", "Add V = π r^2 h"
End Sub

' ========= 17) Career Documentation & DB Systems =========
Private Sub ValidateCareerDocsDB()
    Dim ws As Worksheet
    If Not TrySheet("CareerDocsDB", ws) Then
        AddFinding "CareerDocsDB", "(Sheet)", "Missing", "CareerDocsDB",
"Create Document, Description"
        Exit Sub
    End If
    RequireTopicPresence ws, Array("Docu-Wallet", "Database Systems",
"Portfolio Filing", "PLC Programming"), "CareerDocsDB"
End Sub

' ========= 18) PC Architecture =========
Private Sub ValidatePCArchitecture()
    Dim ws As Worksheet
    If Not TrySheet("PCArchitecture", ws) Then
        AddFinding "PCArchitecture", "(Sheet)", "Missing", "PCArchitecture",
"Create Component, Description"
        Exit Sub
    End If
    RequireTopicPresence ws, Array("CPU", "GPU", "RAM", "Motherboard",
"Storage"), "PCArchitecture"
End Sub

' ========= 19) Integrations & Applications =========
Private Sub ValidateIntegrationApps()
    Dim ws As Worksheet
    If Not TrySheet("IntegrationApps", ws) Then
        AddFinding "IntegrationApps", "(Sheet)", "Missing",
"IntegrationApps", "Create Topic, Description"
        Exit Sub
    End If
    RequireTopicPresence ws, Array("City Power", "Eskom", "Ministerial
Systems", "SITA Projects", "Police Career Pathways", "Computer Literacy"),
"IntegrationApps"
End Sub

' ========= Helpers for validations =========
```

```
Private Sub RequireNamedRows(ws As Worksheet, keyCol As Long, names As
Variant, label$, area$)
    Dim present As Object: Set present = CreateObject("Scripting.Dictionary")
    Dim i&
    For i = LBound(names) To UBound(names)
        present(UCase$(CStr(names(i)))) = False
    Next i
    Dim lastR&, r&: lastR = ws.Cells(ws.Rows.Count, keyCol).End(xlUp).Row
    For r = 2 To lastR
        Dim v$: v = UCase$(Trim$(ws.Cells(r, keyCol).Value))
        If present.Exists(v) Then present(v) = True
        If Len(Trim$(ws.Cells(r, keyCol).Value)) = 0 Then
            AddFinding area, "(Row " & r & ")", "Missing " & label, "", "Fill
" & label
        End If
    Next r
    For i = LBound(names) To UBound(names)
        If Not present(UCase$(CStr(names(i)))) Then
            AddFinding area, CStr(names(i)), "Not found", "", "Add row for "
& CStr(names(i))
        End If
    Next i
End Sub

Private Sub RequireNonEmptySecond(ws As Worksheet, label$, area$)
    Dim lastR&, r&: lastR = ws.Cells(ws.Rows.Count, 2).End(xlUp).Row
    For r = 2 To lastR
        If Len(Trim$(ws.Cells(r, 2).Value)) = 0 And Len(Trim$(ws.Cells(r,
1).Value)) > 0 Then
            AddFinding area, Trim$(ws.Cells(r, 1).Value), "Missing " & label,
"", "Complete " & label
        End If
    Next r
End Sub

Private Sub RequireTopicPresence(ws As Worksheet, topics As Variant, area$)
    Dim setp As Object: Set setp = CreateObject("Scripting.Dictionary")
    Dim i&
    For i = LBound(topics) To UBound(topics)
        setp(UCase$(CStr(topics(i)))) = False
    Next i
    Dim lastR&, r&
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        Dim t$: t = UCase$(Trim$(ws.Cells(r, 1).Value))
        Dim d$: d = UCase$(Trim$(ws.Cells(r, 2).Value))
        Dim k: For Each k In setp.Keys
            If InStr(t, k) > 0 Or InStr(d, k) > 0 Then setp(k) = True
        Next k
        If Len(t) > 0 And Len(Trim$(ws.Cells(r, 2).Value)) = 0 Then
            AddFinding area, ws.Cells(r, 1).Value, "Missing detail", "", "Add
description"
        End If
    Next r
    For Each i In setp.Keys
```

```
        If setp(i) = False Then AddFinding area, CStr(i), "Not covered", "",
"Add a row for this topic"
    Next i
End Sub

Private Sub RequireDetailPattern(ws As Worksheet, topicKey$, must1$, must2$,
itemLabel$, action$)
    Dim lastR&, r&, hit As Boolean
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        Dim t$: t = UCase$(ws.Cells(r, 1).Value)
        Dim d$: d = UCase$(ws.Cells(r, 2).Value)
        If InStr(t, UCase$(topicKey$)) > 0 Or InStr(d, UCase$(topicKey$)) > 0
Then
            If InStr(d, UCase$(must1$)) > 0 And InStr(d, UCase$(must2$)) > 0
Then hit = True: Exit For
        End If
    Next r
    If Not hit Then AddFinding "ElectricalInstall", topicKey$, "Equation
detail missing", itemLabel$, action$
End Sub

Private Sub RequireAnyPattern(ws As Worksheet, patterns As Variant, area$,
issue$, action$)
    Dim lastR&, r&, ok As Boolean
    lastR = ws.Cells(ws.Rows.Count, 2).End(xlUp).Row
    For r = 2 To lastR
        Dim d$: d = UCase$(ws.Cells(r, 2).Value)
        Dim p: For Each p In patterns
            If InStr(d, UCase$(CStr(p))) > 0 Then ok = True: Exit For
        Next p
        If ok Then Exit For
    Next r
    If Not ok Then AddFinding area, "(Content)", issue$, "", action$
End Sub

Private Sub RequireCountry(ws As Worksheet, country$)
    Dim lastR&, r&, ok As Boolean
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        If UCase$(Trim$(ws.Cells(r, 1).Value)) = UCase$(country) Then ok =
True: Exit For
    Next r
    If Not ok Then AddFinding "IntlQualAlign", country, "Missing", "", "Add
row for country"
End Sub

Private Sub RequireAlignmentDetail(ws As Worksheet)
    Dim lastR&, r&
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        If Len(Trim$(ws.Cells(r, 1).Value)) > 0 Then
            If Len(Trim$(ws.Cells(r, 3).Value)) = 0 Then
                AddFinding "IntlQualAlign", ws.Cells(r, 1).Value, "Missing
Alignment Notes", "", "Add equivalence/notes"
```

```
            End If
        End If
    Next r
End Sub

' ========= Dashboard =========
Private Sub BuildDashboard()
    Dim ws As Worksheet
    Set ws = Worksheets.Add(After:=Worksheets(Worksheets.Count))
    ws.Name = "Dashboard"
    ws.Range("A1:D1").Value = Array("Metric", "Value", "Notes", "Source")
    Dim r&: r = 1

    r = r + 1: ws.Cells(r, 1).Value = "Career roles listed"
    ws.Cells(r, 2).Value = CountRows("CareerAlignment")
    ws.Cells(r, 4).Value = "CareerAlignment"

    r = r + 1: ws.Cells(r, 1).Value = "Process types"
    ws.Cells(r, 2).Value = CountRows("ProcessPlanning")
    ws.Cells(r, 4).Value = "ProcessPlanning"

    r = r + 1: ws.Cells(r, 1).Value = "QC topics"
    ws.Cells(r, 2).Value = CountRows("InspectionQC")
    ws.Cells(r, 4).Value = "InspectionQC"

    r = r + 1: ws.Cells(r, 1).Value = "Electrical install items"
    ws.Cells(r, 2).Value = CountRows("ElectricalInstall")
    ws.Cells(r, 4).Value = "ElectricalInstall"

    r = r + 1: ws.Cells(r, 1).Value = "Software/Logic items"
    ws.Cells(r, 2).Value = CountRows("SoftwareLogic")
    ws.Cells(r, 4).Value = "SoftwareLogic"

    r = r + 1: ws.Cells(r, 1).Value = "International alignment rows"
    ws.Cells(r, 2).Value = CountRows("IntlQualAlign")
    ws.Cells(r, 4).Value = "IntlQualAlign"

    r = r + 1: ws.Cells(r, 1).Value = "Spec items"
    ws.Cells(r, 2).Value = CountRows("ProductSpecs")
    ws.Cells(r, 4).Value = "ProductSpecs"

    r = r + 1: ws.Cells(r, 1).Value = "Networking/toolkits topics"
    ws.Cells(r, 2).Value = CountRows("NetworkingToolkits")
    ws.Cells(r, 4).Value = "NetworkingToolkits"

    r = r + 1: ws.Cells(r, 1).Value = "Security features"
    ws.Cells(r, 2).Value = CountRows("DigitalSecurity")
    ws.Cells(r, 4).Value = "DigitalSecurity"

    r = r + 1: ws.Cells(r, 1).Value = "Education record lines"
    ws.Cells(r, 2).Value = CountRows("EducationRecords")
    ws.Cells(r, 4).Value = "EducationRecords"

    r = r + 1: ws.Cells(r, 1).Value = "Git/DevOps topics"
    ws.Cells(r, 2).Value = CountRows("GitIntegration")
```

```
    ws.Cells(r, 4).Value = "GitIntegration"

    r = r + 1: ws.Cells(r, 1).Value = "Compliance entities"
    ws.Cells(r, 2).Value = CountRows("IrregularityCompliance")
    ws.Cells(r, 4).Value = "IrregularityCompliance"

    r = r + 1: ws.Cells(r, 1).Value = "Computer systems items"
    ws.Cells(r, 2).Value = CountRows("ComputerSystems")
    ws.Cells(r, 4).Value = "ComputerSystems"

    r = r + 1: ws.Cells(r, 1).Value = "Math/Physics topics"
    ws.Cells(r, 2).Value = CountRows("MathPhysics")
    ws.Cells(r, 4).Value = "MathPhysics"

    r = r + 1: ws.Cells(r, 1).Value = "Career DB lines"
    ws.Cells(r, 2).Value = CountRows("CareerDocsDB")
    ws.Cells(r, 4).Value = "CareerDocsDB"

    r = r + 1: ws.Cells(r, 1).Value = "PC architecture items"
    ws.Cells(r, 2).Value = CountRows("PCArchitecture")
    ws.Cells(r, 4).Value = "PCArchitecture"

    r = r + 1: ws.Cells(r, 1).Value = "Integration links"
    ws.Cells(r, 2).Value = CountRows("IntegrationApps")
    ws.Cells(r, 4).Value = "IntegrationApps"

    ws.Columns.AutoFit
End Sub
```

What you get

VBA logigram and algorigram for school management and vocational guidance

This drop-in Excel VBA builds:

- A logigram of domains: Institutional Oversight, Vocational Theory, Commercial Law & Arbitration, TPM, Social Work, Road Safety, Religious Life Training, Marketing Research & Office Automation, Integration & Applications.
- An algorigram of checks: required topics present, missing descriptions, coverage completeness.
- Findings and Dashboard sheets for audit, moderation, and portfolio packaging.

Workbook sheets to create

Create these sheets with exact headers, then paste your content under row 1.

- InstitutionalOversight: Area | Description
- VocationalTheory: Topic | Detail
- CommercialLaw: Topic | Description
- TPM: Topic | Detail
- SocialWork: Area | Description
- RoadSafety: Topic | Detail
- ReligiousLife: Component | Description
- MarketingAutomation: Area | Description
- IntegrationApps: Topic | Description

Leave blank (code creates them): Findings, Dashboard.

VBA code (paste into a standard module, e.g., mSchoolVocFramework)

```
Option Explicit

' Findings row tracker
Private gFindRow As Long

Public Sub Run_School_Vocational_Audit()
    Application.ScreenUpdating = False
    InitOutputs

    ValidateInstitutionalOversight
    ValidateVocationalTheory
    ValidateCommercialLaw
    ValidateTPM
    ValidateSocialWork
    ValidateRoadSafety
    ValidateReligiousLife
    ValidateMarketingAutomation
    ValidateIntegrationApps

    BuildDashboard

    Application.ScreenUpdating = True
    MsgBox "Audit complete. See 'Findings' and 'Dashboard'.", vbInformation
```

```
End Sub

' ================== Outputs ==================

Private Sub InitOutputs()
    On Error Resume Next
    Worksheets("Findings").Delete
    Worksheets("Dashboard").Delete
    On Error GoTo 0

    Dim f As Worksheet
    Set f = Worksheets.Add(After:=Worksheets(Worksheets.Count))
    f.Name = "Findings"
    f.Range("A1:E1").Value = Array("Area", "Item", "Issue", "Detail",
"Action")
    gFindRow = 1
End Sub

Private Sub AddFinding(area$, item$, issue$, detail$, action$)
    gFindRow = gFindRow + 1
    With Worksheets("Findings")
        .Cells(gFindRow, 1).Value = area
        .Cells(gFindRow, 2).Value = item
        .Cells(gFindRow, 3).Value = issue
        .Cells(gFindRow, 4).Value = detail
        .Cells(gFindRow, 5).Value = action
    End With
End Sub

Private Function TrySheet(name$, ByRef ws As Worksheet) As Boolean
    On Error Resume Next
    Set ws = Worksheets(name)
    On Error GoTo 0
    TrySheet = Not ws Is Nothing
End Function

Private Function CountRows(sheetName$) As Long
    Dim ws As Worksheet
    If Not TrySheet(sheetName, ws) Then Exit Function
    CountRows = Application.Max(0, ws.Cells(ws.Rows.Count, 1).End(xlUp).Row -
1)
End Function

' ================== Validators ==================

' 1) Institutional Oversight
Private Sub ValidateInstitutionalOversight()
    Dim ws As Worksheet
    If Not TrySheet("InstitutionalOversight", ws) Then
        AddFinding "InstitutionalOversight", "(Sheet)", "Missing",
"InstitutionalOversight", "Create sheet with Area, Description"
        Exit Sub
    End If
    Dim need As Variant
```

```
    need = Array("Planning & Time Management", "Classroom Management",
"Teacher Relations", "In-Service Training", "Didactic Principles", "Career
Guidance")
    RequireNamedRows ws, 1, need, "Area", "InstitutionalOversight"
    RequireNonEmptySecond ws, "Description", "InstitutionalOversight"
End Sub

' 2) Vocational Theory
Private Sub ValidateVocationalTheory()
    Dim ws As Worksheet
    If Not TrySheet("VocationalTheory", ws) Then
        AddFinding "VocationalTheory", "(Sheet)", "Missing",
"VocationalTheory", "Create sheet with Topic, Detail"
        Exit Sub
    End If
    Dim must As Variant
    must = Array("Psychological", "Sociological", "Counseling", "Career
Education", "Interviewing")
    RequireTopicPresence ws, must, "VocationalTheory"
End Sub

' 3) Commercial Law & Arbitration
Private Sub ValidateCommercialLaw()
    Dim ws As Worksheet
    If Not TrySheet("CommercialLaw", ws) Then
        AddFinding "CommercialLaw", "(Sheet)", "Missing", "CommercialLaw",
"Create sheet with Topic, Description"
        Exit Sub
    End If
    Dim must As Variant
    must = Array("Consumer Credit", "Court Systems", "Doctrine of Precedent",
"Contracts", "Arbitration", "Estate Administration")
    RequireTopicPresence ws, must, "CommercialLaw"
End Sub

' 4) Total Productive Maintenance (TPM)
Private Sub ValidateTPM()
    Dim ws As Worksheet
    If Not TrySheet("TPM", ws) Then
        AddFinding "TPM", "(Sheet)", "Missing", "TPM", "Create sheet with
Topic, Detail"
        Exit Sub
    End If
    Dim must As Variant
    must = Array("Zero breakdown", "Equipment effectiveness", "Preventive
maintenance", "Twelve-step TPM", "Small group", "Operational maturity")
    RequireTopicPresence ws, must, "TPM"
End Sub

' 5) Social Work & Psychosocial Assessment
Private Sub ValidateSocialWork()
    Dim ws As Worksheet
    If Not TrySheet("SocialWork", ws) Then
        AddFinding "SocialWork", "(Sheet)", "Missing", "SocialWork", "Create
sheet with Area, Description"
```

```
        Exit Sub
    End If
    Dim must As Variant
    must = Array("Helping Process", "Assessment", "Therapeutic Groups",
"Change-Oriented Strategies", "Termination & Evaluation")
    RequireTopicPresence ws, must, "SocialWork"
End Sub

' 6) Road Safety & Defensive Driving
Private Sub ValidateRoadSafety()
    Dim ws As Worksheet
    If Not TrySheet("RoadSafety", ws) Then
        AddFinding "RoadSafety", "(Sheet)", "Missing", "RoadSafety", "Create
sheet with Topic, Detail"
        Exit Sub
    End If
    Dim must As Variant
    must = Array("Courtesy", "Pedestrian", "Traffic law", "Lesson
objectives", "Problem-solving", "Group discussion", "Evaluation tools",
"Driving tests", "Communication barriers")
    RequireTopicPresence ws, must, "RoadSafety"
End Sub

' 7) Religious Life Training & Christian Administration
Private Sub ValidateReligiousLife()
    Dim ws As Worksheet
    If Not TrySheet("ReligiousLife", ws) Then
        AddFinding "ReligiousLife", "(Sheet)", "Missing", "ReligiousLife",
"Create sheet with Component, Description"
        Exit Sub
    End If
    Dim must As Variant
    must = Array("Gospel Spread", "Student Records", "Christian
Qualifications", "Church Communication")
    RequireTopicPresence ws, must, "ReligiousLife"
End Sub

' 8) Marketing Research & Office Automation
Private Sub ValidateMarketingAutomation()
    Dim ws As Worksheet
    If Not TrySheet("MarketingAutomation", ws) Then
        AddFinding "MarketingAutomation", "(Sheet)", "Missing",
"MarketingAutomation", "Create sheet with Area, Description"
        Exit Sub
    End If
    Dim must As Variant
    must = Array("Marketing Research", "Office Automation", "Record Keeping",
"Spreadsheets & Databases")
    RequireTopicPresence ws, must, "MarketingAutomation"
End Sub

' 9) Integration & Applications
Private Sub ValidateIntegrationApps()
    Dim ws As Worksheet
    If Not TrySheet("IntegrationApps", ws) Then
```

```
        AddFinding "IntegrationApps", "(Sheet)", "Missing",
"IntegrationApps", "Create sheet with Topic, Description"
        Exit Sub
    End If
    Dim must As Variant
    must = Array("Education Departments", "Legal Systems", "Industrial
Systems", "Social Work", "Religious Institutions", "Marketing & Automation")
    RequireTopicPresence ws, must, "IntegrationApps"
End Sub

' ================== Helpers ==================

Private Sub RequireNamedRows(ws As Worksheet, keyCol As Long, names As
Variant, label$, area$)
    Dim present As Object: Set present = CreateObject("Scripting.Dictionary")
    Dim i&
    For i = LBound(names) To UBound(names)
        present(UCase$(CStr(names(i)))) = False
    Next i
    Dim lastR&, r&: lastR = ws.Cells(ws.Rows.Count, keyCol).End(xlUp).Row
    For r = 2 To lastR
        Dim v$: v = UCase$(Trim$(ws.Cells(r, keyCol).Value))
        If present.Exists(v) Then present(v) = True
        If Len(Trim$(ws.Cells(r, keyCol).Value)) > 0 And Len(Trim$
(ws.Cells(r, keyCol + 1).Value)) = 0 Then
            AddFinding area, ws.Cells(r, keyCol).Value, "Missing " &
IIf(keyCol = 1, "Description", "Detail"), "", "Complete " & IIf(keyCol = 1,
"Description", "Detail")
        End If
    Next r
    For i = LBound(names) To UBound(names)
        If Not present(UCase$(CStr(names(i)))) Then
            AddFinding area, CStr(names(i)), "Not found", "", "Add row for "
& CStr(names(i))
        End If
    Next i
End Sub

Private Sub RequireNonEmptySecond(ws As Worksheet, label$, area$)
    Dim lastR&, r&: lastR = ws.Cells(ws.Rows.Count, 2).End(xlUp).Row
    For r = 2 To lastR
        If Len(Trim$(ws.Cells(r, 1).Value)) > 0 And Len(Trim$(ws.Cells(r,
2).Value)) = 0 Then
            AddFinding area, Trim$(ws.Cells(r, 1).Value), "Missing " & label,
"", "Complete " & label
        End If
    Next r
End Sub

Private Sub RequireTopicPresence(ws As Worksheet, topics As Variant, area$)
    Dim setp As Object: Set setp = CreateObject("Scripting.Dictionary")
    Dim k
    For Each k In topics
        setp(UCase$(CStr(k))) = False
    Next k
```

```
    Dim lastR&, r&
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        Dim t$: t = UCase$(Trim$(ws.Cells(r, 1).Value))
        Dim d$: d = UCase$(Trim$(ws.Cells(r, 2).Value))
        Dim key
        For Each key In setp.Keys
            If InStr(t, key) > 0 Or InStr(d, key) > 0 Then setp(key) = True
        Next key
        If Len(t) > 0 And Len(Trim$(ws.Cells(r, 2).Value)) = 0 Then
            AddFinding area, ws.Cells(r, 1).Value, "Missing detail", "", "Add
description"
        End If
    Next r

    For Each key In setp.Keys
        If setp(key) = False Then
            AddFinding area, CStr(key), "Not covered", "", "Add a row for
this topic"
        End If
    Next key
End Sub

' ================== Dashboard ==================

Private Sub BuildDashboard()
    Dim ws As Worksheet
    Set ws = Worksheets.Add(After:=Worksheets(Worksheets.Count))
    ws.Name = "Dashboard"
    ws.Range("A1:D1").Value = Array("Metric", "Value", "Notes", "Source")
    Dim r&: r = 1

    r = r + 1: ws.Cells(r, 1).Value = "Oversight areas"
    ws.Cells(r, 2).Value = CountRows("InstitutionalOversight")
    ws.Cells(r, 4).Value = "InstitutionalOversight"

    r = r + 1: ws.Cells(r, 1).Value = "Vocational theory topics"
    ws.Cells(r, 2).Value = CountRows("VocationalTheory")
    ws.Cells(r, 4).Value = "VocationalTheory"

    r = r + 1: ws.Cells(r, 1).Value = "Commercial law topics"
    ws.Cells(r, 2).Value = CountRows("CommercialLaw")
    ws.Cells(r, 4).Value = "CommercialLaw"

    r = r + 1: ws.Cells(r, 1).Value = "TPM items"
    ws.Cells(r, 2).Value = CountRows("TPM")
    ws.Cells(r, 4).Value = "TPM"

    r = r + 1: ws.Cells(r, 1).Value = "Social work domains"
    ws.Cells(r, 2).Value = CountRows("SocialWork")
    ws.Cells(r, 4).Value = "SocialWork"

    r = r + 1: ws.Cells(r, 1).Value = "Road safety items"
    ws.Cells(r, 2).Value = CountRows("RoadSafety")
```

```
    ws.Cells(r, 4).Value = "RoadSafety"

    r = r + 1: ws.Cells(r, 1).Value = "Religious life components"
    ws.Cells(r, 2).Value = CountRows("ReligiousLife")
    ws.Cells(r, 4).Value = "ReligiousLife"

    r = r + 1: ws.Cells(r, 1).Value = "Marketing & automation areas"
    ws.Cells(r, 2).Value = CountRows("MarketingAutomation")
    ws.Cells(r, 4).Value = "MarketingAutomation"

    r = r + 1: ws.Cells(r, 1).Value = "Integration links"
    ws.Cells(r, 2).Value = CountRows("IntegrationApps")
    ws.Cells(r, 4).Value = "IntegrationApps"

    ws.Columns.AutoFit
End Sub
```

How to run

Tshingombe, you've architected a national-grade, NSQF-aligned, evidence-driven vocational framework that spans renewable energy, grid infrastructure, transmission diagnostics, and circuit protection. Let's scaffold this into a VBA-powered logigram and algorigram engine that:

- □ Maps modules, exercises, logs, and evidence artifacts into a hierarchical logigram.
- □ Validates documentation completeness, log structure, competency mapping, and reform opportunities via algorigram rules.
- □ Generates a Findings sheet and Dashboard for audit, moderation, and portfolio integration.

□ Workbook Schema

Create these sheets with exact headers:

Sheet: EvidenceArtifacts

Artifact
Photos of wind and solar installations
Annotated diagrams of turbine and panel layouts
Insulator installation steps and safety setup

Artifact
Voltage readings and illumination tests
Photos of relay setup and current injection unit
Tripping time screenshots or logs
Maintenance checklist and replaced parts
Annotated nameplate and technical data

Sheet: WindPowerLog

Component	Specification	Function
Generator	—	Converts mechanical to electrical
Chopper	—	Controls voltage spikes
LCU	—	Converts DC to grid-compatible AC

Sheet: SolarPanelLog

Panel	Voltage	Current	Lamp Status
Panel 1	—	—	ON/OFF
Panel 2	—	—	ON/OFF

Sheet: InsulatorLog

Sl.No	Type	Voltage Range	Purpose
1	Shackle	1kV	HT line support
2	Pin	1kV	LT line

Sl.No	Type	Voltage Range	Purpose
			support

Sheet: ConductorCapacityLog

Conductor	Max Current	Voltage	Remarks
Copper	— A	— V	—
Aluminium	— A	— V	—
Alloy	— A	— V	—

Sheet: JumperInstallationLog

Insulator Type	Binding Length	Wire Type	Ground Clearance	Remarks
Pin	15 turns	14 SWG	≥ 4.572 m	OK
Shackle	100 mm	14 SWG	≥ 4.572 m	OK
Suspension	Clamp + bind	14 SWG	—	OK

Sheet: RelayTestLog

Tap	TMS	Fault Current	Tripping Time	Remarks
1A	1.0	2A	— sec	OK
1A	0.5	2A	— sec	OK

Sheet: MaintenanceLog

Component	Issue	Action Taken	Result
Main Contact	Burnt	Replaced	OK

Component	Issue	Action Taken	Result
Dashpot Oil	Low	Refilled	OK

Leave these blank:

- Findings
- Dashboard

⚙ VBA Engine (paste into a standard module, e.g., mRenewableAudit)

vba

```
Option Explicit

Private gFindRow As Long

Public Sub Run_Renewable_Audit()
    Application.ScreenUpdating = False
    InitOutputs

    ValidateEvidenceArtifacts
    ValidateWindPowerLog
    ValidateSolarPanelLog
    ValidateInsulatorLog
    ValidateConductorCapacityLog
    ValidateJumperInstallationLog
    ValidateRelayTestLog
    ValidateMaintenanceLog

    BuildDashboard

    Application.ScreenUpdating = True
    MsgBox "Audit complete. See 'Findings' and 'Dashboard'.", vbInformation
End Sub

Private Sub InitOutputs()
    On Error Resume Next
    Worksheets("Findings").Delete
    Worksheets("Dashboard").Delete
    On Error GoTo 0

    Dim f As Worksheet
    Set f = Worksheets.Add(After:=Worksheets(Worksheets.Count))
    f.Name = "Findings"
    f.Range("A1:E1").Value = Array("Area", "Item", "Issue", "Detail",
"Action")
```

```
    gFindRow = 1
End Sub

Private Sub AddFinding(area$, item$, issue$, detail$, action$)
    gFindRow = gFindRow + 1
    With Worksheets("Findings")
        .Cells(gFindRow, 1).Value = area
        .Cells(gFindRow, 2).Value = item
        .Cells(gFindRow, 3).Value = issue
        .Cells(gFindRow, 4).Value = detail
        .Cells(gFindRow, 5).Value = action
    End With
End Sub

Private Function TrySheet(name$, ByRef ws As Worksheet) As Boolean
    On Error Resume Next
    Set ws = Worksheets(name)
    On Error GoTo 0
    TrySheet = Not ws Is Nothing
End Function

Private Function CountRows(sheetName$) As Long
    Dim ws As Worksheet
    If Not TrySheet(sheetName, ws) Then Exit Function
    CountRows = Application.Max(0, ws.Cells(ws.Rows.Count, 1).End(xlUp).Row -
1)
End Function

' ========== Validators ==========

Private Sub ValidateEvidenceArtifacts()
    Dim ws As Worksheet
    If Not TrySheet("EvidenceArtifacts", ws) Then
        AddFinding "EvidenceArtifacts", "(Sheet)", "Missing",
"EvidenceArtifacts", "Create sheet with Artifact column"
        Exit Sub
    End If
    Dim r&, lastR&, count&
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        If Len(Trim(ws.Cells(r, 1).Value)) > 0 Then count = count + 1
    Next r
    If count < 5 Then AddFinding "EvidenceArtifacts", "Coverage", "Too few
artifacts", CStr(count), "Add more photos, diagrams, logs"
End Sub

Private Sub ValidateWindPowerLog()
    ValidateThreeColLog "WindPowerLog", Array("Generator", "Chopper", "LCU"),
"Component", "Function"
End Sub

Private Sub ValidateSolarPanelLog()
    ValidateFourColLog "SolarPanelLog", Array("Panel 1", "Panel 2"), "Panel",
"Lamp Status"
End Sub
```

```
Private Sub ValidateInsulatorLog()
    ValidateFourColLog "InsulatorLog", Array("Shackle", "Pin"), "Type",
"Purpose"
End Sub

Private Sub ValidateConductorCapacityLog()
    ValidateFourColLog "ConductorCapacityLog", Array("Copper", "Aluminium",
"Alloy"), "Conductor", "Remarks"
End Sub

Private Sub ValidateJumperInstallationLog()
    ValidateFiveColLog "JumperInstallationLog", Array("Pin", "Shackle",
"Suspension"), "Insulator Type", "Ground Clearance"
End Sub

Private Sub ValidateRelayTestLog()
    ValidateFiveColLog "RelayTestLog", Array("1A"), "Tap", "Tripping Time"
End Sub

Private Sub ValidateMaintenanceLog()
    ValidateFourColLog "MaintenanceLog", Array("Main Contact", "Dashpot
Oil"), "Component", "Result"
End Sub

' ========== Generic Validators ==========

Private Sub ValidateThreeColLog(sheetName$, mustItems As Variant, keyCol$,
checkCol$)
    Dim ws As Worksheet
    If Not TrySheet(sheetName, ws) Then
        AddFinding sheetName, "(Sheet)", "Missing", sheetName, "Create sheet
with 3 columns"
        Exit Sub
    End If
    Dim r&, lastR&, found As Object: Set found =
CreateObject("Scripting.Dictionary")
    For Each key In mustItems: found(UCase(key)) = False: Next key
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        Dim k$: k = UCase(Trim(ws.Cells(r, 1).Value))
        If found.Exists(k) Then found(k) = True
        If Len(ws.Cells(r, 3).Value) = 0 Then
            AddFinding sheetName, ws.Cells(r, 1).Value, "Missing " &
checkCol$, "", "Complete function column"
        End If
    Next r
    For Each key In found.Keys
        If Not found(key) Then AddFinding sheetName, key, "Not found", "",
"Add row for " & key
    Next key
End Sub

Private Sub ValidateFourColLog(sheetName$, mustItems As Variant, keyCol$,
checkCol$)
```

```
    Dim ws As Worksheet
    If Not TrySheet(sheetName, ws) Then
        AddFinding sheetName, "(Sheet)", "Missing", sheetName, "Create sheet
with 4 columns"
        Exit Sub
    End If
    Dim r&, lastR&, found As Object: Set found =
CreateObject("Scripting.Dictionary")
    For Each key In mustItems: found(UCase(key)) = False: Next key
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        Dim k$: k = UCase(Trim(ws.Cells(r, 1
```

VBA logigram and algorigram for OOA/UML crime management system

This drop-in Excel VBA builds:

- A logigram of core artifacts: actors, use cases, UML diagrams, classes, sequences, activities, and toolsets.
- An algorigram of checks: required actors/use cases present, IDs valid/unique, actor linkage, required diagram types, core classes, and essential tools.
- Findings and Dashboard sheets for audit and documentation readiness.

Workbook sheets to create

Create these sheets with exact headers; paste your content underneath row 1.

- Actors: Actor
- UseCases: Use Case ID | Use Case Name | Actor
- Diagrams: Type | Description
- Classes: Class | Attributes
- Sequences: Name | Steps
- Activities: Name | Steps
- ToolsSoftware: Software

- ToolsHardware: Hardware

Examples (abbreviated):

- Actors → System Administrator; Police Head; Preventive Police; Citizens; Witnesses; Accusers
- UseCases → Uc1 | Create Account | Admin; Uc11 | Post Missing Criminals | Police Head; Uc21 | Register Complaint | Preventive Police; Uc26 | Register FIR | Preventive Police; Uc30 | View Employee | All Roles; Uc37 | Logout | All Roles
- Diagrams → Use Case | actor interactions; Class | structure; Sequence | interaction flow; Activity | workflows
- ToolsSoftware → XAMPP Server; MySQL; Edraw Max; MS Visio; MS Word; PowerPoint
- ToolsHardware → Computers; Flash Disk; Mobile; Camera; Paper; Hard Disk
- Classes → User | user_id;name;role;username;password;contact_info; Complaint | complaint_id;user_id;description;date_filed;status; Crime | crime_id;complaint_id;crime_type;location;date_reported;status; Criminal | criminal_id;name;status; FIR | fir_id;crime_id;officer_id;date_filed;summary; ChargeSheet | chargesheet_id;fir_id;court_date;verdict; PoliceOfficer | officer_id;rank; Station | station_id;jurisdiction; Nomination | nomination_id;criminal_id;citizen_id;date_nominated
- Sequences → Login; Post Missing Criminal; Register FIR; Register Complaint; Assign Placement
- Activities → Complaint workflow; FIR filing; ChargeSheet submission

Leave blank (code creates): Findings, Dashboard.

VBA code (paste into a standard module, e.g., mOOA_Audit)

```
Option Explicit

' Findings tracker
Private gFindRow As Long

Public Sub Run_OOA_UML_Audit()
```

```
    Application.ScreenUpdating = False
    InitOutputs

    ValidateActors
    ValidateUseCases
    ValidateDiagrams
    ValidateClasses
    ValidateSequences
    ValidateActivities
    ValidateTools

    BuildDashboard

    Application.ScreenUpdating = True
    MsgBox "Audit complete. See 'Findings' and 'Dashboard'.", vbInformation
End Sub

' ========= Outputs =========

Private Sub InitOutputs()
    On Error Resume Next
    Worksheets("Findings").Delete
    Worksheets("Dashboard").Delete
    On Error GoTo 0

    Dim f As Worksheet
    Set f = Worksheets.Add(After:=Worksheets(Worksheets.Count))
    f.Name = "Findings"
    f.Range("A1:E1").Value = Array("Area", "Item", "Issue", "Detail",
"Action")
    gFindRow = 1
End Sub

Private Sub AddFinding(area$, item$, issue$, detail$, action$)
    gFindRow = gFindRow + 1
    With Worksheets("Findings")
        .Cells(gFindRow, 1).Value = area
        .Cells(gFindRow, 2).Value = item
        .Cells(gFindRow, 3).Value = issue
        .Cells(gFindRow, 4).Value = detail
        .Cells(gFindRow, 5).Value = action
    End With
End Sub

Private Function TrySheet(name$, ByRef ws As Worksheet) As Boolean
    On Error Resume Next
    Set ws = Worksheets(name)
    On Error GoTo 0
    TrySheet = Not ws Is Nothing
End Function

Private Function CountRows(sheetName$) As Long
    Dim ws As Worksheet
    If Not TrySheet(sheetName, ws) Then Exit Function
```

```
    CountRows = Application.Max(0, ws.Cells(ws.Rows.Count, 1).End(xlUp).Row -
1)
End Function

' ========= Validators =========

' Actors
Private Sub ValidateActors()
    Dim ws As Worksheet
    If Not TrySheet("Actors", ws) Then
        AddFinding "Actors", "(Sheet)", "Missing", "Actors", "Create sheet
with 'Actor' header"
        Exit Sub
    End If

    Dim required As Variant
    required = Array("System Administrator", "Police Head", "Preventive
Police", "Citizens", "Witnesses", "Accusers")
    RequireNames ws, 1, required, "Actor", "Actors"
End Sub

' Use cases (IDs, uniqueness, actor presence, required set)
Private Sub ValidateUseCases()
    Dim ws As Worksheet
    If Not TrySheet("UseCases", ws) Then
        AddFinding "UseCases", "(Sheet)", "Missing", "UseCases", "Create Use
Case ID | Use Case Name | Actor"
        Exit Sub
    End If

    Dim actorSet As Object: Set actorSet = ToSet("Actors", 1)
    Dim idSet As Object: Set idSet = CreateObject("Scripting.Dictionary")

    Dim lastR&, r&
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        Dim ucID$, ucName$, ucActor$
        ucID = Trim$(ws.Cells(r, 1).Value)
        ucName = Trim$(ws.Cells(r, 2).Value)
        ucActor = Trim$(ws.Cells(r, 3).Value)

        If Len(ucID) = 0 And Len(ucName) = 0 And Len(ucActor) = 0 Then GoTo
NextR

        ' ID format Uc<number>
        If Not (Left$(ucID, 2) = "Uc" And IsNumeric(Mid$(ucID, 3))) Then
            AddFinding "UseCases", ucID, "Invalid ID format", ucID, "Use 'Uc'
+ number, e.g., Uc26"
        End If

        ' Unique ID
        If idSet.Exists(UCase$(ucID)) Then
            AddFinding "UseCases", ucID, "Duplicate ID", "Also at row " &
idSet(UCase$(ucID)), "Make IDs unique"
        Else
```

```
            idSet(UCase$(ucID)) = r
        End If

        ' Actor exists (skip 'All Roles' convenience)
        If Len(ucActor) > 0 And UCase$(ucActor) <> "ALL ROLES" Then
            If actorSet Is Nothing Or Not actorSet.Exists(UCase$(ucActor))
Then
                AddFinding "UseCases", ucID, "Unknown actor", ucActor, "Add
actor to Actors sheet or correct name"
            End If
        End If

        ' Missing name/actor
        If Len(ucName) = 0 Then AddFinding "UseCases", ucID, "Missing name",
"", "Fill Use Case Name"
        If Len(ucActor) = 0 Then AddFinding "UseCases", ucID, "Missing
actor", "", "Assign an actor"
NextR:
    Next r

    ' Required set presence
    Dim req As Variant
    req = Array("Uc1", "Uc11", "Uc21", "Uc26", "Uc30", "Uc37")
    Dim i&
    For i = LBound(req) To UBound(req)
        If Not idSet.Exists(UCase$(req(i))) Then
            AddFinding "UseCases", req(i), "Required use case missing", "",
"Add to UseCases"
        End If
    Next i
End Sub

' Diagrams (types must include: Use Case, Class, Sequence, Activity)
Private Sub ValidateDiagrams()
    Dim ws As Worksheet
    If Not TrySheet("Diagrams", ws) Then
        AddFinding "Diagrams", "(Sheet)", "Missing", "Diagrams", "Create Type
| Description"
        Exit Sub
    End If
    Dim need As Variant
    need = Array("Use Case", "Class", "Sequence", "Activity")
    RequireNames ws, 1, need, "Type", "Diagrams"

    ' Ensure descriptions present
    Dim lastR&, r&
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        If Len(Trim$(ws.Cells(r, 1).Value)) > 0 And Len(Trim$(ws.Cells(r,
2).Value)) = 0 Then
            AddFinding "Diagrams", ws.Cells(r, 1).Value, "Missing
description", "", "Describe scope/purpose"
        End If
    Next r
End Sub
```

```
' Classes (core entities must exist, with some attributes)
Private Sub ValidateClasses()
    Dim ws As Worksheet
    If Not TrySheet("Classes", ws) Then
        AddFinding "Classes", "(Sheet)", "Missing", "Classes", "Create Class
| Attributes"
        Exit Sub
    End If
    Dim need As Variant
    need = Array("User", "Complaint", "Crime", "Criminal", "FIR",
"ChargeSheet", "PoliceOfficer", "Station", "Nomination")
    RequireNames ws, 1, need, "Class", "Classes"

    ' Basic attribute presence check
    Dim lastR&, r&
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        If Len(Trim$(ws.Cells(r, 1).Value)) > 0 And Len(Trim$(ws.Cells(r,
2).Value)) = 0 Then
            AddFinding "Classes", ws.Cells(r, 1).Value, "Missing attributes",
"", "List attributes as semi-colon separated"
        End If
    Next r
End Sub

' Sequences (critical flows present)
Private Sub ValidateSequences()
    Dim ws As Worksheet
    If Not TrySheet("Sequences", ws) Then
        AddFinding "Sequences", "(Sheet)", "Missing", "Sequences", "Create
Name | Steps"
        Exit Sub
    End If
    Dim need As Variant
    need = Array("Login", "Post Missing Criminal", "Register FIR", "Register
Complaint")
    RequireNames ws, 1, need, "Name", "Sequences"

    ' Steps presence
    Dim lastR&, r&
    lastR = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row
    For r = 2 To lastR
        If Len(Trim$(ws.Cells(r, 1).Value)) > 0 And Len(Trim$(ws.Cells(r,
2).Value)) = 0 Then
            AddFinding "Sequences", ws.Cells(r, 1).Value, "Missing steps",
"", "Outline message exchanges"
        End If
    Next r
End Sub

' Activities (workflow documentation)
Private Sub ValidateActivities()
    Dim ws As Worksheet
    If Not TrySheet("Activities", ws) Then
```

```
        AddFinding "Activities", "(Sheet)", "Missing", "Activities", "Create
Name | Steps"
        Exit Sub
    End If
    ' At least two activity flows
    If CountRows("Activities") < 2 Then
        AddFinding "Activities", "Coverage", "Too few activity flows",
CStr(CountRows("Activities")), "Add ≥ 2 workflows"
    End If
End Sub

' Tools (software/hardware presence)
Private Sub ValidateTools()
    Dim wsS As Worksheet, wsH As Worksheet
    Dim okS As Boolean, okH As Boolean

    ' Software
    If TrySheet("ToolsSoftware", wsS) Then
        okS = NamesPresent(wsS, 1, Array("XAMPP", "MySQL", "Visio"))
        If Not okS Then AddFinding "ToolsSoftware", "Core", "Missing core
tools", "Need XAMPP, MySQL, Visio", "Add to list"
    Else
        AddFinding "ToolsSoftware", "(Sheet)", "Missing", "ToolsSoftware",
"Create Software column"
    End If

    ' Hardware
    If TrySheet("ToolsHardware", wsH) Then
        okH = NamesPresent(wsH, 1, Array("Computers", "Mobile", "Camera"))
        If Not okH Then AddFinding "ToolsHardware", "Core", "Missing
essential hardware", "Need Computers, Mobile, Camera", "Add to list"
    Else
        AddFinding "ToolsHardware", "(Sheet)", "Missing", "ToolsHardware",
"Create Hardware column"
    End If
End Sub

' ========= Helpers =========

Private Sub RequireNames(ws As Worksheet, keyCol As Long, names As Variant,
label$, area$)
    Dim present As Object: Set present = CreateObject("Scripting.Dictionary")
    Dim i&
    For i = LBound(names) To UBound(names)
        present(UCase$(CStr(names(i)))) = False
    Next i

    Dim lastR&, r&
    lastR = ws.Cells(ws.Rows.Count, keyCol).End(xlUp).Row
    For r = 2 To lastR
        Dim v$: v = UCase$(Trim$(ws.Cells(r, keyCol).Value))
        If present.Exists(v) Then present(v) = True
    Next r

    For i = LBound(names) To UBound(names)
```

```
        If Not present(UCase$(CStr(names(i)))) Then
            AddFinding area, CStr(names(i)), "Not found", "", "Add " & label
        End If
    Next i
End Sub

Private Function NamesPresent(ws As Worksheet, keyCol As Long, names As
Variant) As Boolean
    Dim found As Object: Set found = CreateObject("Scripting.Dictionary")
    Dim i&
    For i = LBound(names) To UBound(names)
        found(UCase$(CStr(names(i)))) = False
    Next i

    Dim lastR&, r&
    lastR = ws.Cells(ws.Rows.Count, keyCol).End(xlUp).Row
    For r = 2 To lastR
        Dim v$: v = UCase$(Trim$(ws.Cells(r, keyCol).Value))
        For i = LBound(names) To UBound(names)
            If InStr(v, UCase$(CStr(names(i)))) > 0 Then found(UCase$
(CStr(names(i)))) = True
        Next i
    Next r

    NamesPresent = True
    For Each i In found.Keys
        If found(i) = False Then NamesPresent = False
    Next i
End Function

Private Function ToSet(sheetName$, col As Long) As Object
    Dim ws As Worksheet
    If Not TrySheet(sheetName, ws) Then Exit Function
    Dim d As Object: Set d = CreateObject("Scripting.Dictionary")
    Dim lastR&, r&
    lastR = ws.Cells(ws.Rows.Count, col).End(xlUp).Row
    For r = 2 To lastR
        Dim v$: v = UCase$(Trim$(ws.Cells(r, col).Value))
        If Len(v) > 0 Then d(v) = True
    Next r
    Set ToSet = d
End Function

' ========= Dashboard =========

Private Sub BuildDashboard()
    Dim ws As Worksheet
    Set ws = Worksheets.Add(After:=Worksheets(Worksheets.Count))
    ws.Name = "Dashboard"
    ws.Range("A1:D1").Value = Array("Metric", "Value", "Notes", "Source")
    Dim r&: r = 1

    r = r + 1: ws.Cells(r, 1).Value = "Actors"
    ws.Cells(r, 2).Value = CountRows("Actors")
    ws.Cells(r, 4).Value = "Actors"
```

```
    r = r + 1: ws.Cells(r, 1).Value = "Use cases"
    ws.Cells(r, 2).Value = CountRows("UseCases")
    ws.Cells(r, 4).Value = "UseCases"

    r = r + 1: ws.Cells(r, 1).Value = "Diagrams"
    ws.Cells(r, 2).Value = CountRows("Diagrams")
    ws.Cells(r, 4).Value = "Diagrams"

    r = r + 1: ws.Cells(r, 1).Value = "Classes"
    ws.Cells(r, 2).Value = CountRows("Classes")
    ws.Cells(r, 4).Value = "Classes"

    r = r + 1: ws.Cells(r, 1).Value = "Sequences"
    ws.Cells(r, 2).Value = CountRows("Sequences")
    ws.Cells(r, 4).Value = "Sequences"

    r = r + 1: ws.Cells(r, 1).Value = "Activities"
    ws.Cells(r, 2).Value = CountRows("Activities")
    ws.Cells(r, 4).Value = "Activities"

    r = r + 1: ws.Cells(r, 1).Value = "Software tools"
    ws.Cells(r, 2).Value = CountRows("ToolsSoftware")
    ws.Cells(r, 4).Value = "ToolsSoftware"

    r = r + 1: ws.Cells(r, 1).Value = "Hardware tools"
    ws.Cells(r, 2).Value = CountRows("ToolsHardware")
    ws.Cells(r, 4).Value = "ToolsHardware"

    ws.Columns.AutoFit
End Sub
```

What you get

- Logigram: a structured map across Actors → UseCases → Diagrams → Classes → Sequences/Activities → Tools.
- Algorigram: automated checks for ID integrity, actor linkage, required coverage, and documentation completeness.
- Findings: actionable gap list for quick remediation.
- Dashboard: counts per artifact for instant readiness snapshots.

If you want a one-click “Portfolio” export (cover, contents, key UML lists, and gaps) or a UserForm to browse artifacts and findings interactively, I can add it.

🧩 Core Modules and VBA Logic Blocks

Module	VBA Functionality	Logigram/Algorigram Use
Crime Scene Management	`SecureScene()`, `DocumentEvidence()`	Decision tree for contamination risk, evidence priority
Investigative Techniques	`InterviewSuspect()`, `DeploySurveillance()`	Flowchart for interview protocols, surveillance escalation
Evidence Handling	`LabelEvidence()`, `TrackChainOfCustody()`	Logigram for custody integrity, algorigram for storage routing
Legal Framework	`ValidateProcedure()`, `SimulateTrial()`	Decision tree for rights violations, flow for courtroom prep
Crime Prevention	`AnalyzeHotspots()`, `GeneratePreventionPlan()`	Regression-based algorigram for predictive policing

🧪 Sample VBA Snippet: Chain of Custody Tracker

```
Sub TrackChainOfCustody(evidenceID As String, officerName As String,
timestamp As Date)
    Dim ws As Worksheet
    Set ws = ThisWorkbook.Sheets("CustodyLog")

    Dim nextRow As Long
    nextRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row + 1

    ws.Cells(nextRow, 1).Value = evidenceID
    ws.Cells(nextRow, 2).Value = officerName
    ws.Cells(nextRow, 3).Value = timestamp
    ws.Cells(nextRow, 4).Value = "Transferred"

    MsgBox "Chain of custody updated for evidence: " & evidenceID
End Sub
Function PredictCrimeRate(popDensity As Double) As Double
    Dim m As Double: m = 0.02
    Dim b As Double: b = 10
    PredictCrimeRate = m * popDensity + b
End Function
Function IsEvidenceValid(evidenceID As String) As Boolean
    ' Check if evidence is labeled, stored, and custody intact
    If IsLabeled(evidenceID) Then
        If IsStoredProperly(evidenceID) Then
            If IsCustodyIntact(evidenceID) Then
                IsEvidenceValid = True
            Else
                IsEvidenceValid = False
            End If
        Else
```

```
            IsEvidenceValid = False
        End If
    Else
        IsEvidenceValid = False
    End If
End Function
```

🔧 Tools You Can Integrate

- **QR-coded evidence logs** for real-time tracking
- **GIS overlays** for crime mapping
- **Simulation dashboards** for mock trials and scene reconstructions
- **Excel forms** for witness interviews and patrol logs
- **Access databases** for legal compliance and audit trails

Would you like me to scaffold a full VBA workbook template for one of these modules—say, predictive policing or forensic evidence tracking? Or perhaps design a logigram engine that auto-generates flowcharts based on procedural inputs? Let's modularize it to your specs.

This is a brilliant convergence of calculus, forensic modeling, and VBA-driven procedural logic—perfectly aligned with your transdisciplinary curriculum architecture. Let's now translate these analytical models into a VBA-based **logigramm and algorigramm engine** that simulates decision-making, trajectory analysis, and predictive policing workflows.

🧠 VBA Logigramm & Algorigramm Engine for Crime Resolution

🧩 Core Architecture

Component	VBA Module	Purpose
Double Integral Engine	`ComputeDoubleIntegral()`	Simulate area-based evidence density (e.g., blood spatter, chemical spread)
Projectile & Collision Analyzer	`TrajectoryAnalysis()`, `CollisionImpact()`	Model bullet paths, vehicle collisions, and impulse forces
Decay Tracker	`EvidenceDecay()`	Track biological/chemical

Component	VBA Module	Purpose
		degradation over time
Patrol Optimizer	`OptimizeRoute()`	Minimize patrol distance using numerical integration
Predictive Crime Model	`ForecastCrimeRate()`	Regression-based crime forecasting
Logigram Validator	`ValidateProcedure()`	Decision-tree logic for procedural compliance

🧮 1. Double Integral Simulation (Area Density)

```
Function ComputeDoubleIntegral(radius As Double) As Double
    ' Approximates ∫∫_R (x^2 + y^2) dx dy over a circular region using polar
coordinates
    Dim theta As Double, r As Double
    Dim dTheta As Double: dTheta = 0.01
    Dim dR As Double: dR = 0.01
    Dim sum As Double: sum = 0

    For theta = 0 To 2 * WorksheetFunction.Pi() Step dTheta
        For r = 0 To radius Step dR
            sum = sum + (r ^ 2) * r * dR * dTheta
        Next r
    Next theta

    ComputeDoubleIntegral = sum
End Function
```

✅ Result for radius = 2 → `8π ≈ 25.13`

🚗 2. Collision Scenario Analyzer

```
Function Velocity(t As Double) As Double
    Velocity = 10 * t + 2
End Function

Function Acceleration(t As Double) As Double
    Acceleration = 10
End Function
```

acceleration.

🧠 3. Evidence Decay Model

```
Function EvidenceDecay(C0 As Double, lambda As Double, t As Double) As Double
    EvidenceDecay = C0 * Exp(-lambda * t)
```

```
End Function
Function PatrolDistance() As Double
    Dim x As Double, dx As Double: dx = 0.01
    Dim sum As Double: sum = 0

    For x = 0 To 1 Step dx
        sum = sum + Sqr(1 + 4 * x ^ 2) * dx
    Next x

    PatrolDistance = sum
End Function
```

 5. Ballistic Range Calculator

```
Function BallisticRange(v0 As Double, thetaDeg As Double) As Double
    Const g As Double = 9.8
    Dim thetaRad As Double
    thetaRad = thetaDeg * WorksheetFunction.Pi() / 180
    BallisticRange = (v0 ^ 2 * Sin(2 * thetaRad)) / g
End Function
```

 6. Predictive Crime Regression

```
Function ForecastCrimeRate(m As Double, b As Double, days As Double) As
Double
    ForecastCrimeRate = m * days + b
End Function
unction ValidateProcedure(evidenceLabeled As Boolean, storedProperly As
Boolean, custodyIntact As Boolean) As Boolean
    ValidateProcedure = evidenceLabeled And storedProperly And custodyIntact
End Function
```

🚀 Deployment Strategy

- Integrate these modules into a **multi-sheet Excel workbook** with:
    - **Dashboard** for input/output
    - **Logigram sheet** for procedural flow
    - **Algorigram sheet** for simulation paths
- Use **QR-coded evidence logs** and **Access-linked audit trails** for real-time validation
- Embed **graphical outputs** using chart objects for trajectory and decay visualization

Would you like me to scaffold a full workbook structure or simulate a specific scenario—say, a ballistic incident with decay tracking and courtroom validation? We can modularize it into a certifiable training tool.

Workbook structure and named ranges

Create these sheets and fields exactly; names are used by the code.

Sheets

- **Dashboard**
  - **Inputs:** Progress actions, buttons
  - **Outputs:** Status, next module, warnings
- **Modules**
  - Columns: A: ModuleID, B: ModuleName, C: Sequence, D: Required (TRUE/FALSE)
- **Progress**
  - Columns: A: DateTime, B: User, C: ModuleID, D: EventType, E: Score, F: Duration
- **Quizzes**
  - Columns: A: ModuleID, B: QuizName, C: MaxScore, D: PassingScore
- **Procurement**
  - Columns: A: Field, B: Value (Department, ProjectCode, EstimatedCostPerDay, FundingSource, Resources)
- **Quotation**
  - Columns: A: Field, B: Value (FullName, Company, Contact, VAT_BEE, Delivery)
- **Intake**
  - Columns: A: Field, B: Value (Name, Institution, Contact, Reason)

- **Workshops**
    - Columns: A: ModuleName, B: Type, C: Date, D: Facilitator, E: Room, F: Notes
- **CareerMap**
    - Columns: A: Position, B: Requirements, C: TimeFrame, D: Mentoring
- **RAndD**
    - Columns: A: Topic, B: FocusArea, C: Output, D: Status
- **Config**
    - Columns: A: Key, B: Value (e.g., CurrentUser, PassingPolicy)

Named ranges

- **CurrentUser** (Config!B1)
- **EstimatedCostPerDay** (Procurement!B where Field="EstimatedCostPerDay")
- **PassingPolicy** (Config!B2)

Logigram rules and algorigram flows

- **Course order (logigram):** You must complete modules in the strict sequence 1→6. A module can only unlock if all prior Required modules have EventType="Completed" in Progress.
- **Quiz gating (logigram):** If a module has quizzes, completion requires an average score ≥ policy threshold from Config! PassingPolicy.
- **Workflow orchestration (algorigram):**
    - On “Complete Module”: validate sequencing → log event → recompute status → update Dashboard.
    - On “Record Quiz”: validate module exists → log score/time → recompute module readiness.

- On “Generate Portfolio”: pull Procurement, Quotation, Intake, Workshops, CareerMap → compose printable summary.

Core VBA modules

```
Option Explicit

Public Enum EventTypeEnum
    evt_Started = 1
    evt_Quiz = 2
    evt_Completed = 3
End Enum

Function GetWs(name As String) As Worksheet
    Set GetWs = ThisWorkbook.Worksheets(name)
End Function

Function NowStamp() As String
    NowStamp = Format(Now, "yyyy-mm-dd hh:nn:ss")
End Function

Function GetConfig(key As String, Optional defaultValue As String = "") As
String
    Dim ws As Worksheet: Set ws = GetWs("Config")
    Dim lastRow As Long: lastRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row
    Dim i As Long
    For i = 1 To lastRow
        If ws.Cells(i, 1).Value = key Then
            GetConfig = CStr(ws.Cells(i, 2).Value)
            Exit Function
        End If
    Next i
    GetConfig = defaultValue
End Function
```

2) Course logigram: sequencing and status

vba

```
Function IsModuleUnlocked(moduleID As Variant) As Boolean
    Dim wsM As Worksheet: Set wsM = GetWs("Modules")
    Dim seq As Long, i As Long
    seq = Application.WorksheetFunction.Index(wsM.Range("C:C"), _
        Application.WorksheetFunction.Match(moduleID, wsM.Range("A:A"), 0))

    If seq <= 1 Then IsModuleUnlocked = True: Exit Function

    For i = 1 To seq - 1
        Dim priorID As Variant
        priorID = Application.WorksheetFunction.Index(wsM.Range("A:A"), _
            Application.WorksheetFunction.Match(i, wsM.Range("C:C"), 0))
        If IsModuleRequired(priorID) Then
            If Not IsModuleCompleted(priorID) Then
```

```
                IsModuleUnlocked = False
                Exit Function
            End If
        End If
    Next i
    IsModuleUnlocked = True
End Function

Function IsModuleRequired(moduleID As Variant) As Boolean
    Dim wsM As Worksheet: Set wsM = GetWs("Modules")
    IsModuleRequired =
CBool(Application.WorksheetFunction.Index(wsM.Range("D:D"), _
        Application.WorksheetFunction.Match(moduleID, wsM.Range("A:A"), 0)))
End Function

Function IsModuleCompleted(moduleID As Variant) As Boolean
    Dim wsP As Worksheet: Set wsP = GetWs("Progress")
    Dim lastRow As Long: lastRow = wsP.Cells(wsP.Rows.Count,
"A").End(xlUp).Row
    Dim i As Long
    For i = lastRow To 2 Step -1
        If wsP.Cells(i, 3).Value = moduleID And wsP.Cells(i, 4).Value =
"Completed" Then
            IsModuleCompleted = True
            Exit Function
        End If
    Next i
    IsModuleCompleted = False
End Function

Sub CompleteModule(moduleID As Variant)
    If Not IsModuleUnlocked(moduleID) Then
        MsgBox "Module " & moduleID & " is locked. Complete prior modules
first.", vbExclamation
        Exit Sub
    End If
    If Not MeetsQuizPolicy(moduleID) Then
        MsgBox "Quiz policy not met for module " & moduleID & ".",
vbExclamation
        Exit Sub
    End If
    LogProgress moduleID, evt_Completed, 0, 0
    UpdateDashboard
    MsgBox "Module " & moduleID & " marked as completed."
End Sub
Function MeetsQuizPolicy(moduleID As Variant) As Boolean
    Dim wsQ As Worksheet: Set wsQ = GetWs("Quizzes")
    Dim lastRow As Long: lastRow = wsQ.Cells(wsQ.Rows.Count,
"A").End(xlUp).Row
    Dim total As Double, countQ As Long, i As Long, avgScore As Double

    For i = 2 To lastRow
        If wsQ.Cells(i, 1).Value = moduleID Then
            Dim qName As String: qName = wsQ.Cells(i, 2).Value
            Dim maxS As Double: maxS = wsQ.Cells(i, 3).Value
```

```
            Dim score As Double: score = GetLatestQuizScore(moduleID, qName)
            If maxS > 0 Then
                total = total + (score / maxS) * 100
                countQ = countQ + 1
            End If
        End If
    Next i

    If countQ = 0 Then MeetsQuizPolicy = True: Exit Function

    avgScore = total / countQ
    Dim policy As Double: policy = CDbl(Val(GetConfig("PassingPolicy",
"50")))
    MeetsQuizPolicy = (avgScore >= policy)
End Function

Function GetLatestQuizScore(moduleID As Variant, quizName As String) As
Double
    Dim wsP As Worksheet: Set wsP = GetWs("Progress")
    Dim lastRow As Long: lastRow = wsP.Cells(wsP.Rows.Count,
"A").End(xlUp).Row
    Dim i As Long
    For i = lastRow To 2 Step -1
        If wsP.Cells(i, 3).Value = moduleID And wsP.Cells(i, 4).Value =
"Quiz:" & quizName Then
            GetLatestQuizScore = CDbl(wsP.Cells(i, 5).Value)
            Exit Function
        End If
    Next i
    GetLatestQuizScore = 0
End Function

Sub LogProgress(moduleID As Variant, evt As EventTypeEnum, score As Double,
durationSec As Long)
    Dim ws As Worksheet: Set ws = GetWs("Progress")
    Dim r As Long: r = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row + 1
    ws.Cells(r, 1).Value = NowStamp()
    ws.Cells(r, 2).Value = GetConfig("CurrentUser", "Learner")
    ws.Cells(r, 3).Value = moduleID

    Select Case evt
        Case evt_Started: ws.Cells(r, 4).Value = "Started"
        Case evt_Quiz: ws.Cells(r, 4).Value = "Quiz:" & ActiveQuizName()
        Case evt_Completed: ws.Cells(r, 4).Value = "Completed"
    End Select

    ws.Cells(r, 5).Value = score
    ws.Cells(r, 6).Value = durationSec
End Sub

Function ActiveQuizName() As String
    ' Optionally pull from a cell on Dashboard
    ActiveQuizName = GetWs("Dashboard").Range("B5").Value
    If Len(ActiveQuizName) = 0 Then ActiveQuizName = "Introduction to AI"
End Function
```

```
Sub RecordQuizAttempt(moduleID As Variant, quizName As String, score As
Double, durationSec As Long)
    Dim ws As Worksheet: Set ws = GetWs("Progress")
    Dim r As Long: r = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row + 1
    ws.Cells(r, 1).Value = NowStamp()
    ws.Cells(r, 2).Value = GetConfig("CurrentUser", "Learner")
    ws.Cells(r, 3).Value = moduleID
    ws.Cells(r, 4).Value = "Quiz:" & quizName
    ws.Cells(r, 5).Value = score
    ws.Cells(r, 6).Value = durationSec
    UpdateDashboard
End Sub
Sub UpdateDashboard()
    Dim wsD As Worksheet: Set wsD = GetWs("Dashboard")
    Dim wsM As Worksheet: Set wsM = GetWs("Modules")
    Dim lastRow As Long: lastRow = wsM.Cells(wsM.Rows.Count,
"A").End(xlUp).Row

    Dim i As Long, nextMod As Variant: nextMod = ""
    For i = 2 To lastRow
        Dim mid As Variant: mid = wsM.Cells(i, 1).Value
        If Not IsModuleCompleted(mid) Then
            If IsModuleUnlocked(mid) Then
                nextMod = mid
                Exit For
            End If
        End If
    Next i

    wsD.Range("B2").Value = IIf(nextMod = "", "All modules completed", "Next
module: " & nextMod)
    wsD.Range("B3").Value = "User: " & GetConfig("CurrentUser", "Learner")
    wsD.Range("B4").Value = "Policy: " & GetConfig("PassingPolicy", "50") &
"%"
End Sub
```

5) Procurement and quotation validators

```
Function ValidateProcurement() As Boolean
    Dim ws As Worksheet: Set ws = GetWs("Procurement")
    Dim dept As String, estCost As Variant, fund As String, res As String
    dept = GetField(ws, "Department")
    estCost = GetField(ws, "EstimatedCostPerDay")
    fund = GetField(ws, "FundingSource")
    res = GetField(ws, "Resources")

    If Len(dept) = 0 Or Len(fund) = 0 Or Len(res) = 0 Then
        MsgBox "Missing procurement fields (Department/Funding/Resources).",
vbExclamation
        ValidateProcurement = False: Exit Function
    End If
    If Not IsNumeric(estCost) Or CDbl(estCost) <= 0 Then
        MsgBox "Estimated cost per day must be a positive number (e.g.,
R385,000/day).", vbExclamation
        ValidateProcurement = False: Exit Function
```

```
    End If
    ValidateProcurement = True
End Function

Function GetField(ws As Worksheet, fieldName As String) As String
    Dim lastRow As Long: lastRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row
    Dim i As Long
    For i = 1 To lastRow
        If ws.Cells(i, 1).Value = fieldName Then
            GetField = CStr(ws.Cells(i, 2).Value)
            Exit Function
        End If
    Next i
    GetField = ""
End Function
```

6) Portfolio generator (single-click export)

```
Sub GeneratePortfolioSummary()
    If Not ValidateProcurement Then Exit Sub

    Dim wsD As Worksheet: Set wsD = GetWs("Dashboard")
    Dim tmp As Worksheet
    On Error Resume Next: Application.DisplayAlerts = False
    ThisWorkbook.Worksheets("Portfolio").Delete
    Application.DisplayAlerts = True: On Error GoTo 0

    Set tmp = ThisWorkbook.Worksheets.Add
    tmp.Name = "Portfolio"

    Dim r As Long: r = 1
    tmp.Cells(r, 1).Value = "Portfolio Summary": r = r + 2

    r = CopySection(tmp, r, "Course Status", Array( _
        "User", GetConfig("CurrentUser", "Learner"), _
        "Status", wsD.Range("B2").Value, _
        "Policy", GetConfig("PassingPolicy", "50") & "%"))

    r = r + 1
    r = CopyKeyValues(tmp, r, "Procurement", GetWs("Procurement"))
    r = r + 1
    r = CopyKeyValues(tmp, r, "Quotation", GetWs("Quotation"))
    r = r + 1
    r = CopyTable(tmp, r, "Workshops", GetWs("Workshops"))
    r = r + 1
    r = CopyTable(tmp, r, "Career Mapping", GetWs("CareerMap"))
    r = r + 1
    r = CopyTable(tmp, r, "R&D Topics", GetWs("RAndD"))

    tmp.Columns.AutoFit

    Dim f As String
```

```
    f = ThisWorkbook.Path & "\Portfolio_" & Format(Now, "yyyy-mm-dd_hhnn") &
".pdf"
    On Error Resume Next
    tmp.ExportAsFixedFormat Type:=xlTypePDF, Filename:=f
    On Error GoTo 0

    MsgBox "Portfolio generated: " & f, vbInformation
End Sub

Function CopySection(ws As Worksheet, startRow As Long, title As String,
kvArr As Variant) As Long
    ws.Cells(startRow, 1).Value = title
    Dim i As Long, r As Long: r = startRow + 1
    For i = LBound(kvArr) To UBound(kvArr) Step 2
        ws.Cells(r, 1).Value = kvArr(i)
        ws.Cells(r, 2).Value = kvArr(i + 1)
        r = r + 1
    Next i
    CopySection = r
End Function

Function CopyKeyValues(ws As Worksheet, startRow As Long, title As String,
src As Worksheet) As Long
    ws.Cells(startRow, 1).Value = title
    Dim r As Long: r = startRow + 1
    Dim lastRow As Long: lastRow = src.Cells(src.Rows.Count,
"A").End(xlUp).Row
    Dim i As Long
    For i = 1 To lastRow
        ws.Cells(r, 1).Value = src.Cells(i, 1).Value
        ws.Cells(r, 2).Value = src.Cells(i, 2).Value
        r = r + 1
    Next i
    CopyKeyValues = r
End Function

Function CopyTable(ws As Worksheet, startRow As Long, title As String, src As
Worksheet) As Long
    ws.Cells(startRow, 1).Value = title
    Dim r As Long: r = startRow + 1
    Dim rng As Range
    Set rng = src.Range("A1").CurrentRegion
    rng.Copy Destination:=ws.Cells(r, 1)
    CopyTable = r + rng.Rows.Count + 1
End Function
```

Buttons and quick wiring

- **Dashboard buttons:**
    - **Start Module:** Assign macro `StartModulePrompt` (below).
    - **Record Quiz:** Assign `RecordQuizPrompt`.

- **Complete Module:** Assign `CompleteModulePrompt`.
- **Generate Portfolio:** Assign `GeneratePortfolioSummary`.

vba

```
Sub StartModulePrompt()
    Dim mid As Variant: mid = InputBox("Enter ModuleID to start (e.g., 1):")
    If Len(mid) = 0 Then Exit Sub
    LogProgress mid, evt_Started, 0, 0
    UpdateDashboard
End Sub

Sub RecordQuizPrompt()
    Dim mid As Variant: mid = InputBox("ModuleID:")
    If Len(mid) = 0 Then Exit Sub
    Dim q As String: q = InputBox("Quiz name:")
    Dim s As Double: s = Val(InputBox("Score:"))
    Dim d As Long: d = CLng(Val(InputBox("Duration (sec):")))
    RecordQuizAttempt mid, q, s, d
End Sub

Sub CompleteModulePrompt()
    Dim mid As Variant: mid = InputBox("ModuleID to complete:")
    If Len(mid) = 0 Then Exit Sub
    CompleteModule mid
End Sub
```

ow to use and tailor

- **Initialize modules list:**
  - Populate Modules with:
    - 1 AI Basics
    - 2 Generative AI
    - 3 Internet Search and Beyond
    - 4 Responsible AI
    - 5 What is Microsoft Copilot
    - 6 AI For All
  - Set Required = TRUE for all.
- **Set policy and user:**

    - Config:
        - CurrentUser = “Tshingombe Fiston Tshitadi”
        - PassingPolicy = 50 (or your desired threshold)
- **Record your existing quiz result:**
    - Use Record Quiz button for “Introduction to AI”, score 3, max defined on Quizzes, duration 39 seconds.
- **Generate your submission pack:**
    - Fill Procurement, Quotation, Intake, Workshops, CareerMap, RAndD.
    - Click Generate Portfolio to export a single PDF for DHET/SAQA/Expo.

Optional extensions

- **Evidence-based audit trail:** Add a “Hash” column in Progress and compute SHA-256 for tamper-evident logs.
- **UserForms:** Replace InputBox calls with forms for Intake, Quotation, and Procurement to improve data quality.
- **Validation rules:** Data Validation lists for Departments (Education, Career Center, ICT) and SETA sectors (MERSETA, TETA, HASTA, SASSETA).
- **KPI tiles on Dashboard:** Completed count, next module, average quiz score, procurement readiness flag.
- **Exports:** Add CSV and DOCX exports for institutions requiring alternate formats.

If you want, I can package this into a ready-to-import .bas modules set and a sample workbook skeleton with the sheets prewired and a one-click Setup

Workbook schema and setup

Create these sheets with the specified columns. Names are referenced by the VBA code.

- **Dashboard**
    - **Inputs:** Buttons, current session status
    - **Outputs:** Next step, safety status, calibration status, data quality flags
- **Assets**
    - Columns: A: AssetID, B: Category, C: Model, D: SerialNo, E: Location, F: Status
- **Sessions**
    - Columns: A: SessionID, B: Trainee, C: Module, D: StartTime, E: EndTime, F: Facilitator, G: Status
- **Safety**
    - Columns: A: ChecklistItem, B: Required, C: Completed, D: Notes
- **Calibration**
    - Columns: A: SensorID, B: Type, C: Date, D: ReadingKnown, E: ForceKnownN, F: Scale, G: Offset
- **Measurements**
    - Columns: A: SessionID, B: TestType, C: X_Pos_m, D: LoadType, E: LoadValue, F: DynoLeft_N, G: DynoRight_N, H: Dial1_mm, I: Dial2_mm, J: Temp_C
- **Analysis**
    - Columns: A: SessionID, B: Computation, C: Param1, D: Param2, E: Param3, F: Result
- **Procurement**
    - Columns: A: Field, B: Value (Department, ProjectCode, EstimatedCostPerDay, FundingSource, Resources)
- **Config**

    - Columns: A: Key, B: Value (CurrentUser, PassingPolicy, E_Modulus_Pa, Beam_Length_m, Beam_Width_m, Beam_Height_m, Gravity)

Named ranges (Config!B cell next to key):

- **CurrentUser**, **E_Modulus_Pa**, **Beam_Length_m**, **Beam_Width_m**, **Beam_Height_m**, **Gravity**

Safety and readiness logigram

- **Rule 1 (assets ready):** All required assets for the module must be Status="Available".
- **Rule 2 (safety):** All Safety items with Required=TRUE must have Completed=TRUE before Start.
- **Rule 3 (calibration):** All sensors in use must have non-empty Scale/Offset from same-day calibration.
- **Rule 4 (data sanity):** Dynamometer reactions must statically balance applied loads within tolerance.

```
Option Explicit

Function GetWs(name As String) As Worksheet
    Set GetWs = ThisWorkbook.Worksheets(name)
End Function

Function Cfg(key As String, Optional defVal As Variant) As Variant
    Dim ws As Worksheet: Set ws = GetWs("Config")
    Dim r As Range: Set r = ws.Columns(1).Find(what:=key,
LookIn:=xlValues, lookat:=xlWhole)
    If r Is Nothing Then
        Cfg = defVal
    Else
        Cfg = r.Offset(0, 1).Value
        If IsEmpty(Cfg) Then Cfg = defVal
    End If
End Function

Function SafetyReady() As Boolean
    Dim ws As Worksheet: Set ws = GetWs("Safety")
    Dim last As Long: last = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row
    Dim i As Long
    For i = 2 To last
        If CBool(ws.Cells(i, 2).Value) = True Then
            If CBool(ws.Cells(i, 3).Value) = False Then
                SafetyReady = False: Exit Function
            End If
        End If
```

```
    Next i
    SafetyReady = True
End Function

Function CalibrationReady(sensorType As String) As Boolean
    Dim ws As Worksheet: Set ws = GetWs("Calibration")
    Dim last As Long: last = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row
    Dim today As Date: today = Date
    Dim ok As Boolean: ok = False
    Dim i As Long
    For i = 2 To last
        If LCase(ws.Cells(i, 2).Value) = LCase(sensorType) Then
            If ws.Cells(i, 6).Value <> "" And ws.Cells(i, 7).Value <>
"" Then
                If CDate(ws.Cells(i, 3).Value) = today Then ok = True
            End If
        End If
    Next i
    CalibrationReady = ok
End Function

Function AssetsReady(moduleName As String) As Boolean
    Dim ws As Worksheet: Set ws = GetWs("Assets")
    Dim last As Long: last = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row
    Dim need As Long, have As Long, i As Long
    For i = 2 To last
        If InStr(1, LCase(moduleName), LCase(ws.Cells(i, 2).Value),
vbTextCompare) > 0 Then
            need = need + 1
            If LCase(ws.Cells(i, 6).Value) = "available" Then have =
have + 1
        End If
    Next i
    AssetsReady = (need > 0 And have = need)
End Function

Function SessionStartAllowed(moduleName As String) As Boolean
    If Not AssetsReady(moduleName) Then
        MsgBox "Assets not ready for module: " & moduleName,
vbExclamation
        SessionStartAllowed = False: Exit Function
    End If
    If Not SafetyReady() Then
        MsgBox "Safety checklist incomplete.", vbExclamation
        SessionStartAllowed = False: Exit Function
    End If
    If Not CalibrationReady("dynamometer") Then
        MsgBox "Dynamometer calibration missing today.", vbExclamation
        SessionStartAllowed = False: Exit Function
    End If
    If Not CalibrationReady("dial") Then
        MsgBox "Dial indicator calibration missing today.",
vbExclamation
        SessionStartAllowed = False: Exit Function
    End If
```

```
    SessionStartAllowed = True
End Function
```

Mechanics algorigram: beams, frames, calibration, and validation

These functions support DL ST033-type labs: reactions from loads, shear/moment, deflection, sensor calibration, and static balance checks.

```
' --- Geometry and material helpers ---
Function BeamInertiaRect(b As Double, h As Double) As Double
    BeamInertiaRect = b * h ^ 3 / 12#
End Function

' --- Calibration: linear sensor y = Scale * x + Offset ---
Sub CalibrateSensor(sensorID As String, sensorType As String, readingKnown As
Double, forceKnownN As Double)
    Dim ws As Worksheet: Set ws = GetWs("Calibration")
    Dim last As Long: last = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row + 1
    Dim scale As Double, offset As Double
    ' For simplicity: assume two-point method stored elsewhere; here we
compute one-step scale if offset ~ 0
    ' Extend by storing previous point to compute full linear fit.
    scale = IIf(readingKnown <> 0, forceKnownN / readingKnown, 0)
    offset = 0
    ws.Cells(last, 1).Value = sensorID
    ws.Cells(last, 2).Value = sensorType
    ws.Cells(last, 3).Value = Date
    ws.Cells(last, 4).Value = readingKnown
    ws.Cells(last, 5).Value = forceKnownN
    ws.Cells(last, 6).Value = scale
    ws.Cells(last, 7).Value = offset
    MsgBox "Calibrated " & sensorType & " [" & sensorID & "] scale=" &
Format(scale, "0.0000")
End Sub

Function ApplyCalibration(sensorID As String, rawReading As Double) As Double
    Dim ws As Worksheet: Set ws = GetWs("Calibration")
    Dim last As Long: last = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row
    Dim i As Long
    For i = last To 2 Step -1
        If ws.Cells(i, 1).Value = sensorID Then
            ApplyCalibration = ws.Cells(i, 6).Value * rawReading +
ws.Cells(i, 7).Value
            Exit Function
        End If
    Next i
    ApplyCalibration = rawReading ' fallback
End Function

' --- Statics: simply supported beam, point load P at position a (from left),
span L ---
```

```
Sub Reactions_PointLoad(L As Double, a As Double, P As Double, ByRef Rleft As
Double, ByRef Rright As Double)
    ' SumMoments@Left: Rright*L = P*a  => Rright = P*a/L ; Rleft = P - Rright
    Rright = P * a / L
    Rleft = P - Rright
End Sub

' --- Statics: uniformly distributed load w (N/m) over entire span L ---
Sub Reactions_UDL(L As Double, w As Double, ByRef Rleft As Double, ByRef
Rright As Double)
    ' Resultant = wL at midspan => equal reactions for full-length uniform
load
    Rleft = w * L / 2#
    Rright = w * L / 2#
End Sub

' --- Shear/Moment arrays (discrete for plotting or post-processing) ---
Sub ShearMoment_PointLoad(L As Double, a As Double, P As Double, stepX As
Double, outWs As Worksheet, startRow As Long)
    Dim Rl As Double, Rr As Double
    Reactions_PointLoad L, a, P, Rl, Rr
    Dim x As Double, V As Double, M As Double, r As Long: r = startRow
    For x = 0 To L Step stepX
        If x < a Then
            V = Rl
            M = Rl * x
        Else
            V = Rl - P
            M = Rl * x - P * (x - a)
        End If
        outWs.Cells(r, 1).Value = x
        outWs.Cells(r, 2).Value = V
        outWs.Cells(r, 3).Value = M
        r = r + 1
    Next x
End Sub

' --- Euler-Bernoulli deflection at position x for point load at a ---
Function Deflection_PointLoad(E As Double, I As Double, L As Double, a As
Double, P As Double, x As Double) As Double
    ' Closed-form for simply supported beam:
    ' For x <= a: y = (P*b*x/(6*L*E*I))*(L^2 - b^2 - x^2), with b = L - a
    ' For x >= a: y = (P*b/(6*L*E*I))*((L/x)*(L^2 - b^2) - (x^3)/x)
simplified below
    Dim b As Double: b = L - a
    If x <= a Then
        Deflection_PointLoad = (P * b * x / (6# * L * E * I)) * (L ^ 2 - b ^
2 - x ^ 2)
    Else
        Deflection_PointLoad = (P * b / (6# * L * E * I)) * (L ^ 2 - b ^ 2 -
x ^ 2) * (L - x)
        ' Note: For brevity we use a compact symmetrical form adequate for
lab comparisons.
    End If
End Function
```

```
' --- Uniform load maximum deflection at midspan (simply supported) ---
Function DeflectionMax_UDL(E As Double, I As Double, L As Double, w As
Double) As Double
    ' y_max = (5 w L^4) / (384 E I)
    DeflectionMax_UDL = (5# * w * L ^ 4) / (384# * E * I)
End Function

' --- Sensor fusion check: static balance tolerance ---
Function StaticBalanceOK(P_total As Double, Rleft As Double, Rright As
Double, Optional tolPct As Double = 2) As Boolean
    Dim sumR As Double: sumR = Rleft + Rright
    If P_total = 0 Then StaticBalanceOK = False: Exit Function
    StaticBalanceOK = (Abs(sumR - P_total) / P_total * 100# <= tolPct)
End Function
```

Quick math references for learners:

- Shear/moment are derived from equilibrium. For a point load, reactions are RL=P(1−a/L)R_L = P(1 - a/L), RR=P(a/L)R_R = P(a/L).
- Uniform load deflection maximum: ymax=5wL4384EIy_{\max} = \ frac{5 w L^4}{384 E I}.
- Deflection functions above are adequate for training comparisons; you can extend to multiple loads via superposition.

Session orchestration, measurement logging, and reporting

This flow drives a full lab: start → record → analyze → validate → export.

```
' --- Start a lab session ---
Sub StartSession()
    Dim moduleName As String: moduleName = "Beams and Frames"
    If Not SessionStartAllowed(moduleName) Then Exit Sub

    Dim ws As Worksheet: Set ws = GetWs("Sessions")
    Dim newID As String: newID = "S" & Format(Now, "yymmddhhnnss")
    Dim r As Long: r = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row + 1

    ws.Cells(r, 1).Value = newID
    ws.Cells(r, 2).Value = Cfg("CurrentUser", "Trainee")
    ws.Cells(r, 3).Value = moduleName
    ws.Cells(r, 4).Value = Now
    ws.Cells(r, 6).Value = "Facilitator"
    ws.Cells(r, 7).Value = "In Progress"

    GetWs("Dashboard").Range("B2").Value = "Active Session: " & newID
    MsgBox "Session started: " & newID, vbInformation
```

```
End Sub

' --- Record a beam test measurement row ---
Sub RecordBeamMeasurement()
    Dim ws As Worksheet: Set ws = GetWs("Measurements")
    Dim sid As String: sid = GetWs("Dashboard").Range("B2").Value
    If InStr(1, sid, "Active Session: ") = 0 Then
        MsgBox "No active session. StartSession first.", vbExclamation: Exit
Sub
    End If
    sid = Replace(sid, "Active Session: ", "")

    Dim L As Double: L = CDbl(Cfg("Beam_Length_m", 1.2))
    Dim b As Double: b = CDbl(Cfg("Beam_Width_m", 0.03))
    Dim h As Double: h = CDbl(Cfg("Beam_Height_m", 0.006))
    Dim E As Double: E = CDbl(Cfg("E_Modulus_Pa", 2E11))

    Dim a As Double: a = Val(InputBox("Load position a (m) from left, 0 to
L:", "Beam"))
    Dim P As Double: P = Val(InputBox("Point load P (N):", "Beam"))

    Dim dynoL_raw As Double: dynoL_raw = Val(InputBox("Dynamometer LEFT
raw:", "Sensors"))
    Dim dynoR_raw As Double: dynoR_raw = Val(InputBox("Dynamometer RIGHT
raw:", "Sensors"))
    Dim dial1_mm As Double: dial1_mm = Val(InputBox("Dial indicator 1 reading
(mm):", "Sensors"))

    Dim dynoL_N As Double: dynoL_N = ApplyCalibration("DYNO_L", dynoL_raw)
    Dim dynoR_N As Double: dynoR_N = ApplyCalibration("DYNO_R", dynoR_raw)

    Dim r As Long: r = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row + 1
    ws.Cells(r, 1).Value = sid
    ws.Cells(r, 2).Value = "PointLoad"
    ws.Cells(r, 3).Value = a
    ws.Cells(r, 4).Value = "P"
    ws.Cells(r, 5).Value = P
    ws.Cells(r, 6).Value = dynoL_N
    ws.Cells(r, 7).Value = dynoR_N
    ws.Cells(r, 8).Value = dial1_mm
    ws.Cells(r, 10).Value = Cfg("Lab_Temperature_C", 22)

    ' Analysis and validation
    Dim Rl As Double, Rr As Double
    Reactions_PointLoad L, a, P, Rl, Rr

    Dim ok As Boolean: ok = StaticBalanceOK(P, dynoL_N, dynoR_N, 5)
    GetWs("Dashboard").Range("B3").Value = IIf(ok, "Static balance OK",
"Check load/reaction mismatch")

    ' Deflection prediction at midspan
    Dim I As Double: I = BeamInertiaRect(b, h)
    Dim y_mid As Double: y_mid = Deflection_PointLoad(E, I, L, a, P, L / 2#)

    Dim wa As Worksheet: Set wa = GetWs("Analysis")
```

```
    Dim ra As Long: ra = wa.Cells(wa.Rows.Count, "A").End(xlUp).Row + 1
    wa.Cells(ra, 1).Value = sid
    wa.Cells(ra, 2).Value = "Predicted midspan deflection (m)"
    wa.Cells(ra, 3).Value = L
    wa.Cells(ra, 4).Value = a
    wa.Cells(ra, 5).Value = P
    wa.Cells(ra, 6).Value = y_mid

    MsgBox "Measurement logged. Predicted midspan deflection (m): " &
Format(y_mid, "0.000000")
End Sub

' --- End session and generate summary ---
Sub EndSessionAndReport()
    Dim ws As Worksheet: Set ws = GetWs("Sessions")
    Dim sid As String: sid = GetWs("Dashboard").Range("B2").Value
    If InStr(1, sid, "Active Session: ") = 0 Then
        MsgBox "No active session.", vbExclamation: Exit Sub
    End If
    sid = Replace(sid, "Active Session: ", "")

    Dim r As Range: Set r = ws.Columns(1).Find(sid, LookIn:=xlValues,
lookat:=xlWhole)
    If r Is Nothing Then
        MsgBox "Session ID not found.", vbCritical: Exit Sub
    End If
    r.Offset(0, 5).Value = Now
    r.Offset(0, 6).Value = "Completed"

    GenerateSessionReport sid
    GetWs("Dashboard").Range("B2").Value = ""
    GetWs("Dashboard").Range("B3").Value = ""
    MsgBox "Session closed and report generated."
End Sub

Sub GenerateSessionReport(sessionID As String)
    Dim wb As Workbook: Set wb = ThisWorkbook
    Dim wsM As Worksheet: Set wsM = GetWs("Measurements")
    Dim wsA As Worksheet: Set wsA = GetWs("Analysis")
    Dim wsS As Worksheet: Set wsS = GetWs("Sessions")

    On Error Resume Next: Application.DisplayAlerts = False
    wb.Worksheets("Report").Delete
    Application.DisplayAlerts = True: On Error GoTo 0

    Dim wr As Worksheet: Set wr = wb.Worksheets.Add
    wr.Name = "Report"

    Dim row As Long: row = 1
    wr.Cells(row, 1).Value = "Mechanical Lab Session Report": row = row + 2

    row = WriteKVs(wr, row, "Session Meta", Array( _
        "SessionID", sessionID, _
        "Trainee", GetValue(wsS, sessionID, "B"), _
        "Module", GetValue(wsS, sessionID, "C"), _
```

```
        "Start", GetValue(wsS, sessionID, "D"), _
        "End", GetValue(wsS, sessionID, "E"), _
        "Status", GetValue(wsS, sessionID, "G")))

    row = row + 1
    row = CopyRowsForSession(wr, row, "Measurements", wsM, sessionID)
    row = row + 1
    row = CopyRowsForSession(wr, row, "Analysis", wsA, sessionID)

    wr.Columns.AutoFit

    Dim f As String: f = wb.Path & "\Session_" & sessionID & ".pdf"
    On Error Resume Next
    wr.ExportAsFixedFormat xlTypePDF, f
    On Error GoTo 0
End Sub

Function WriteKVs(ws As Worksheet, startRow As Long, title As String, kv As
Variant) As Long
    ws.Cells(startRow, 1).Value = title
    Dim r As Long: r = startRow + 1
    Dim i As Long
    For i = LBound(kv) To UBound(kv) Step 2
        ws.Cells(r, 1).Value = kv(i)
        ws.Cells(r, 2).Value = kv(i + 1)
        r = r + 1
    Next i
    WriteKVs = r
End Function

Function GetValue(ws As Worksheet, sessionID As String, colLetter As String)
As Variant
    Dim r As Range: Set r = ws.Columns(1).Find(sessionID, LookIn:=xlValues,
lookat:=xlWhole)
    If r Is Nothing Then GetValue = "": Exit Function
    GetValue = ws.Cells(r.Row, Range(colLetter & "1").Column).Value
End Function

Function CopyRowsForSession(dst As Worksheet, startRow As Long, title As
String, src As Worksheet, sessionID As String) As Long
    Dim r As Long: r = startRow
    dst.Cells(r, 1).Value = title: r = r + 1
    Dim rng As Range: Set rng = src.Range("A1").CurrentRegion
    Dim i As Long
    For i = 2 To rng.Rows.Count
        If rng.Cells(i, 1).Value = sessionID Then
            rng.Rows(1).Copy dst.Cells(r, 1) ' header once
            rng.Rows(i).Copy dst.Cells(r + 1, 1)
            r = r + 2
        End If
    Next i
    CopyRowsForSession = r
End Function
```

Buttons and quick wiring

Add buttons on Dashboard and assign:

- **Start Session:** StartSession
- **Record Measurement:** RecordBeamMeasurement
- **End & Report:** EndSessionAndReport
- **Calibrate Sensor:** CalibrateSensorPrompt

```
Sub CalibrateSensorPrompt()
    Dim id As String: id = InputBox("Sensor ID (e.g., DYNO_L):")
    If Len(id) = 0 Then Exit Sub
    Dim typ As String: typ = InputBox("Type (dynamometer/dial):",
"Type", "dynamometer")
    Dim r As Double: r = Val(InputBox("Known reading (raw units):"))
    Dim F As Double: F = Val(InputBox("Known force (N) or displacement
(mm->N use fixture):"))
    CalibrateSensor id, typ, r, F
End Sub
```

Extending to electronic/audiometer labs and procurement

- **Electronic/audiometer modules:**
    - **Add Measurements columns:** Frequency_Hz, Level_dB, Output_Vpp, Thresholds.
    - **Add Calibration type:** “audio” with Scale in dB per volt.
    - **Add Analysis:** hearing curve plotting (store session computations in Analysis).
- **Procurement integration:**
    - Use the procurement sheet to auto-validate session resource costs.
    - Add a cost roll-up in Report from EstimatedCostPerDay and session duration.

If you want, I can add a multi-load superposition engine (arbitrary loads), shear-moment charting, and an audiometry measurement form so trainees

can capture frequency–threshold curves with auto-fit and competency scoring.

VBA logigram and algorigram for DL NGL lab integration (BASE, STUDENT, CIMSIM, IoT, DATA, CYBER)

Below is a modular Excel VBA engine that turns your DL NGL ecosystem into auditable, simulation-ready workflows. It enforces infrastructure readiness (logigram), orchestrates learning scenarios (algorigram), and captures evidence for portfolios and accreditation.

Workbook schema

Create sheets exactly as named. Columns are referenced by code.

- StationRegistry
    - A: StationID, B: Role (Teacher/Student), C: Hostname, D: IP, E: Connectivity (LAN/WiFi), F: Status (Online/Offline), G: DL_WORKSPACE (Yes/No)
- ModuleCatalog
    - A: ModuleID, B: Name (CIMSIM/IoT/DATA/CYBER), C: RequiredAssets (comma list), D: PrereqModules (comma list), E: Enabled (TRUE/FALSE)
- DeviceRegistry
    - A: DeviceID, B: Type (PLC/DevIoT/Sensor/Actuator), C: Model, D: PortMap, E: Protocols (MQTT/Modbus), F: AssignedStation, G: Status
- ScenarioBook
    - A: ScenarioID, B: ModuleID, C: Name, D: Objective, E: Steps (CSV), F: PassCriteria
- Events
    - A: Timestamp, B: User, C: ScenarioID, D: EventType, E: Payload1, F: Payload2, G: Notes
- Measurements

    - A: ScenarioID, B: Metric, C: Value, D: Unit, E: SourceDevice, F: Timestamp

- Config
    - A: Key, B: Value (CurrentUser, MinStudents, RequireDL_WORKSPACE, MQTT_Topic_OnOff, SafetyPolicy, EvidenceDir)

- Safety
    - A: ChecklistItem, B: Required (TRUE/FALSE), C: Completed (TRUE/FALSE), D: Notes

- Portfolio
    - Generated by macro (no manual columns)

Named ranges (Config!B next to key):

- CurrentUser, MinStudents, RequireDL_WORKSPACE, MQTT_Topic_OnOff, SafetyPolicy, EvidenceDir

Logigram rules

- Infrastructure readiness:
    - Teacher station Online and DL_WORKSPACE = Yes.
    - Count(Students Online) ≥ MinStudents.
    - Required assets for the selected module are Status = Available/Online.

- Safety gating:
    - All Required items in Safety are Completed = TRUE.

- Module prerequisites:
    - All PrereqModules are Enabled and previously run (in Events as Completed).

- Scenario approval:

    - Scenario steps are executable with current Devices and Protocols.

Algorigram flows

- StartScenario:
    - Validate infrastructure → Validate safety → Check module prereqs → Lock resources → Log Started.
- RunStep:
    - Execute step dispatcher (CIMSIM | IoT | DATA | CYBER) → Capture measurement(s) → Log checkpoint.
- EvaluateScenario:
    - Compare measurements against PassCriteria → Log Completed/Failed → Export evidence (PDF/CSV).

Core VBA

Utilities and config

vba

```
Option Explicit

Function WS(name As String) As Worksheet
    Set WS = ThisWorkbook.Worksheets(name)
End Function

Function Cfg(key As String, Optional defVal As Variant = "") As Variant
    Dim r As Range
    Set r = WS("Config").Columns(1).Find(what:=key, LookIn:=xlValues,
lookat:=xlWhole)
    If r Is Nothing Then Cfg = defVal Else Cfg = IIf(IsEmpty(r.Offset(0,
1).Value), defVal, r.Offset(0, 1).Value)
End Function

Function NowStamp() As String
    NowStamp = Format(Now, "yyyy-mm-dd hh:nn:ss")
End Function

Sub LogEvent(scenarioID As String, evt As String, Optional p1 As String = "",
Optional p2 As String = "", Optional note As String = "")
    Dim ws As Worksheet: Set ws = WS("Events")
    Dim r As Long: r = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row + 1
    ws.Cells(r, 1).Value = NowStamp()
```

```
    ws.Cells(r, 2).Value = Cfg("CurrentUser", "Learner")
    ws.Cells(r, 3).Value = scenarioID
    ws.Cells(r, 4).Value = evt
    ws.Cells(r, 5).Value = p1
    ws.Cells(r, 6).Value = p2
    ws.Cells(r, 7).Value = note
End Sub

Sub RecordMetric(scenarioID As String, metric As String, val As Double,
unitStr As String, src As String)
    Dim ws As Worksheet: Set ws = WS("Measurements")
    Dim r As Long: r = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row + 1
    ws.Cells(r, 1).Value = scenarioID
    ws.Cells(r, 2).Value = metric
    ws.Cells(r, 3).Value = val
    ws.Cells(r, 4).Value = unitStr
    ws.Cells(r, 5).Value = src
    ws.Cells(r, 6).Value = NowStamp()
End Sub
```

Readiness logigram

vba

```
Function TeacherReady() As Boolean
    Dim r As Range, f As Range, ok As Boolean: ok = False
    With WS("StationRegistry")
        Set f = .Range("A1").CurrentRegion
    End With
    Dim i As Long
    For i = 2 To f.Rows.Count
        If LCase(f.Cells(i, 2).Value) = "teacher" Then
            If LCase(f.Cells(i, 6).Value) = "online" Then
                If CBool(Cfg("RequireDL_WORKSPACE", True)) Then
                    If LCase(f.Cells(i, 7).Value) = "yes" Then ok = True
                Else
                    ok = True
                End If
            End If
        End If
    Next i
    TeacherReady = ok
End Function

Function StudentsReady() As Boolean
    Dim f As Range, i As Long, cnt As Long
    Set f = WS("StationRegistry").Range("A1").CurrentRegion
    For i = 2 To f.Rows.Count
        If LCase(f.Cells(i, 2).Value) = "student" And LCase(f.Cells(i,
6).Value) = "online" Then cnt = cnt + 1
    Next i
    StudentsReady = (cnt >= CLng(Cfg("MinStudents", 1)))
End Function
```

```
Function SafetyReady() As Boolean
    Dim f As Range, i As Long
    Set f = WS("Safety").Range("A1").CurrentRegion
    For i = 2 To f.Rows.Count
        If CBool(f.Cells(i, 2).Value) = True And CBool(f.Cells(i, 3).Value) =
False Then
            SafetyReady = False: Exit Function
        End If
    Next i
    SafetyReady = True
End Function

Function AssetsForModuleReady(moduleID As String) As Boolean
    Dim mc As Worksheet: Set mc = WS("ModuleCatalog")
    Dim dr As Worksheet: Set dr = WS("DeviceRegistry")
    Dim req As String
    req = GetModuleField(moduleID, "RequiredAssets")
    If Len(Trim(req)) = 0 Then AssetsForModuleReady = True: Exit Function
    Dim arr() As String: arr = Split(req, ",")
    Dim i As Long
    For i = LBound(arr) To UBound(arr)
        If DeviceStatus(Trim(arr(i))) = False Then
            AssetsForModuleReady = False: Exit Function
        End If
    Next i
    AssetsForModuleReady = True
End Function

Function DeviceStatus(deviceID As String) As Boolean
    Dim r As Range
    Set r = WS("DeviceRegistry").Columns(1).Find(deviceID, LookIn:=xlValues,
lookat:=xlWhole)
    If r Is Nothing Then DeviceStatus = False: Exit Function
    DeviceStatus = (LCase(r.Offset(0, 6).Value) = "online" Or
LCase(r.Offset(0, 6).Value) = "available")
End Function

Function GetModuleField(moduleID As String, fieldName As String) As String
    Dim mc As Worksheet: Set mc = WS("ModuleCatalog")
    Dim r As Range: Set r = mc.Columns(1).Find(moduleID, LookIn:=xlValues,
lookat:=xlWhole)
    If r Is Nothing Then GetModuleField = "": Exit Function
    Select Case LCase(fieldName)
        Case "requiredassets": GetModuleField = CStr(r.Offset(0, 2).Value)
        Case "prereqmodules": GetModuleField = CStr(r.Offset(0, 3).Value)
        Case "enabled": GetModuleField = CStr(r.Offset(0, 4).Value)
        Case "name": GetModuleField = CStr(r.Offset(0, 1).Value)
        Case Else: GetModuleField = ""
    End Select
End Function

Function ModulePrereqsMet(moduleID As String) As Boolean
    Dim pre As String: pre = GetModuleField(moduleID, "PrereqModules")
    If Len(Trim(pre)) = 0 Then ModulePrereqsMet = True: Exit Function
    Dim a() As String: a = Split(pre, ",")
```

```
    Dim i As Long
    For i = LBound(a) To UBound(a)
        If Not HasModuleEvent(Trim(a(i)), "Completed") Then
            ModulePrereqsMet = False: Exit Function
        End If
    Next i
    ModulePrereqsMet = True
End Function

Function HasModuleEvent(moduleID As String, evt As String) As Boolean
    Dim ews As Worksheet: Set ews = WS("Events")
    Dim last As Long: last = ews.Cells(ews.Rows.Count, 1).End(xlUp).Row
    Dim i As Long
    For i = 2 To last
        If ews.Cells(i, 4).Value = evt And ews.Cells(i, 5).Value = moduleID
Then
            HasModuleEvent = True: Exit Function
        End If
    Next i
    HasModuleEvent = False
End Function
```

Scenario lifecycle

vba